15

OpenFlow/SDN and Optical Networks

Contents

Service Provider Optical Networks 388
Optical Transport Network 389
Optical Network Management and Intelligent Control Plane 391
How Can SDN/OpenFlow Improve Optical Network Control? 393
  Goals of Applying SDN/OpenFlow to Optical Networks 393
Potential Applications for SDN/OpenFlow in Optical Networks 394
  Packet-Optical Interconnection (POI) 394
  Data Center Interconnection and Network Virtualization 395
  Private Optical Networks 396
Extending OpenFlow/SDN to Optical Networks 397
  What Are the Challenges to OpenFlow/SDN for Optical Networks? 397
    OpenFlow Extensions for Circuits 397
    Optical Impairments 398
    Lack of Visibility into the Data Stream 398
    Technology Specific Discovery 398
    Service Provider Policy and Interoperability 399
    Reliability and Scalability 399
  Research on OpenFlow/SDN for Optical Networks 400
    Ciena/Stanford Prototyping 400
    OpenFlow in Europe—Linking Infrastructure and Applications (OFELIA) Project 402
How Will OpenFlow/SDN Be Deployed in Carrier Optical Networks? 404
  OpenFlow/SDN Overlay with Transport Tunnels for POI 404
  OpenFlow Overlay Over the Control Plane (with Abstraction) 405
  Direct OpenFlow Control of Packet/Optical 406

Standards and Interoperability 407
Open Networking Foundation 407
IETF Standards 408
PCE and Stateful PCE 409
Optical Internetworking Forum (OIF) 410
Conclusions for the Future 410
References 411

Service Provider Optical Networks

International Telecommunications Union – Telecommunication Standardization Sector (ITU-T) Recommendation G.805 [1], a core standard for service provider network architecture, defines transport network as "the functional resources of the network, which conveys user information between locations." Optical transport networks (OTNs) provide the underlying connectivity in service provider networks, allowing information to be conveyed between central office locations across metro areas, long distances, and undersea networks.

Optical fiber links typically support the wavelength of multiple optical channels (OCh) multiplexed on a dense wavelength division multiplexing (DWDM) transmission system. DWDM systems carry 80 to 100 wavelengths on a fiber pair, where each wavelength carries a single high-rate signal, such as 100-Gb/s Ethernet, or may carry a multiplex of lower rate signals. For example, a 100-Gb/s channel may be composed of 10 component signals, each of which is a 10-Gb/s Ethernet signal. Services provided by optical networks include the transport of packetized IP and Ethernet traffic and private line services such as an Ethernet private line or private wavelength services.

Optical transport can be used as point-to-point links connecting large packet switches, mesh or ring topology networks incorporating photonic switches or add/drop multiplexers (optical-optical-optical or O-O-O), or mesh networks incorporating electronic time division multiplexing (TDM) switching systems (optical-electronic-optical or O-E-O), depending on the services being offered.

Optical wavelength services offered by service providers can vary in bandwidth from the capacity of a wavelength to a small fraction of that amount. Currently, several carriers offer 40-G services as the high end of the spectrum. The low end is typically 50 to 155 Mbps

using synchronous optical network (SONET) or synchronous digital hierarchy (SDH) standards. Some of the fastest growing optical services today are 1 G and 10 G because of the growth of the Ethernet [2]. Optical network services are typically not highly dynamic; only in a few cases have carriers deployed optical network services that are under some dynamic customer control [3]. The promise of new services and applications that could make use of dynamic optical networking is a chief driver for carrier interest in OpenFlow/software-defined network (SDN).

Optical Transport Network

The current ITU-T standards define an optical transport hierarchy (OTH) [4] for optical network multiplexing and switching. Networks built according to these standards are called OTNs. OTN includes two fundamentally different types of optical switching technologies: photonic switching of wavelengths, which are called OCh, without termination of the wavelength; and electrical switching of the digital components of the terminated wavelength, which are called OCh data units (ODUs). OTN is also used as a framing format to provide interoperability, performance monitoring, and management of a signal even without switching. OTN replaces SONET or SDH standards that are still the majority of optical switches and services deployed today, allowing channel rates of 40 Gb/s and higher.

OTN is organized into a series of layers incorporating both photonic and electronic signals.

In Figure 15.1, the layers from OCh downward are photonic, whereas the layers from OTU upward are electronic/digital. Each layer incorporates its own overhead (OH) for performance monitoring, bit-oriented signaling, etc. Forward error correction (FEC) is added before the signal enters the optical domain to improve performance. The architecture for OTN is defined in ITU-T Recommendation G.872 [5], and the format for OTN, in Recommendation G.709-v1 [6]. Several digital container rates are defined in OTN: in its initial version, ODU1 (2.5 Gbps), ODU2 (10 Gbps), and ODU3 (40 Gbps), subsequently extended by ITU-T Recommendation G.709-v3 to introduce new containers for GbE (ODU0) and 100 GbE (ODU4).

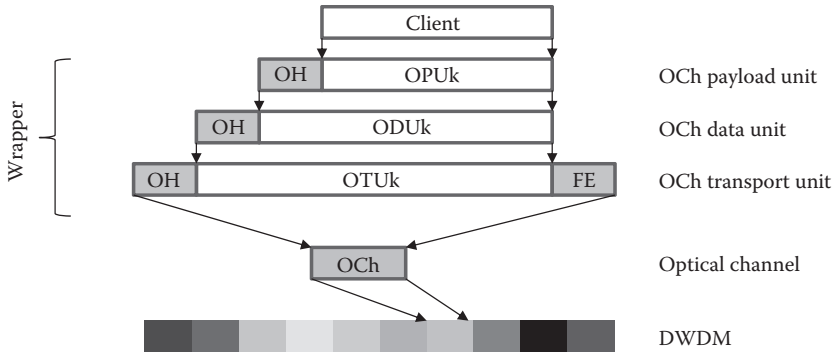


Figure 15.1 Layering of electronic and photonic signals (FE, forward error correction).

OTN electrical switches support up to 4 Tb/s of nonblocking switching capacity in a full-height equipment chassis. Multichassis configurations allow switches to scale to 100 Tb/s and beyond if necessary. Digital monitoring points support rapid fault identification and localization. Electrical switching allows for the easy addition and extraction of signals from the transmission system, with a variety of service protection/restoration capabilities using fast electrical reconfiguration.

OTN photonic switches, usually referred to as reconfigurable optical add/drop multiplexers (ROADMs), support the express routing of an OCh through a network node to avoid the additional cost and energy consumption of electrical switching, but with more constraints.

The combination of electrical and photonic switching components is used in today's optical network to support global scalability balanced against cost; electrical switching allows signal regeneration to increase distance while removing optical impairments, but with additional circuitry and energy usage. Embedded management OH information in the digital signal supports performance monitoring and forms the basis for end-to-end service assurance.

Recent trends in optical network technology include the following:

- The use of tunable transmitters and receivers as components that allow more dynamic control over the wavelength to be used to carry the signal over a particular port
- The development of coherent receivers and variable FEC algorithms to increase signal data rates over the basic 12.5-GHz ITU grid to up to 100 Gbps, with improved signal detection ability over fibers of different characteristics

- The introduction of colorless, directionless, and contentionless ROADM designs that (for a price) reduce port-to-port connectivity and wavelength constraints previously encountered with ROADMs [7]
- The introduction of a flexible grid structure for an optical bandwidth that potentially allows variable spectrum allocation for a higher signal bandwidth or longer distances traveled

Optical Network Management and Intelligent Control Plane

For many years, optical networks were managed using central management systems, which required manual intervention through graphic user interfaces (GUIs) to carry out changes in configuration. The management system provided the database of equipment and components, and each network element (NE) provided a management interface that was used by the management system to control cross-connection.

Beginning approximately 10 years ago, distributed control plane protocols were introduced in the control architecture of optical networks as a way to improve the accuracy of network databases, the speed of provisioning, and the efficiency of recovery. Distributed automatic control of optical networks has resulted in significant reductions in network operations cost; an increase in network availability to 1 defect per million resulting from multiple stages of recovery; and enabled new service offerings, notable of which are customer-controlled optical network services [9].

The distributed control plane is now incorporated into many carrier optical networks because of its ability to automate management functions and support self-healing in response to failures. The control plane has been particularly successful in core networks, where there is a higher degree of connectivity, and in undersea networks, where repairs to failed links is difficult and expensive and the ability to automatically recover from multiple failures is highly valuable. The control plane has mainly been used for the electrical layers of transport networks; the use for most photonic networks is impractical because of the added complexity of path computation for all-optical or photonic links and the greater latency involved in reconfiguring photonic components [10].

In distributed control, control plane messages are exchanged over the signaling control network (SCN), which may consist of in-band signaling links such as the data communication channel (DCC) and general communication channel (GCC) in SONET/SDH and OTN, respectively, and the optical supervisory channel at the OTN optical layer, or use a dedicated out-of-band packet network connecting NEs.

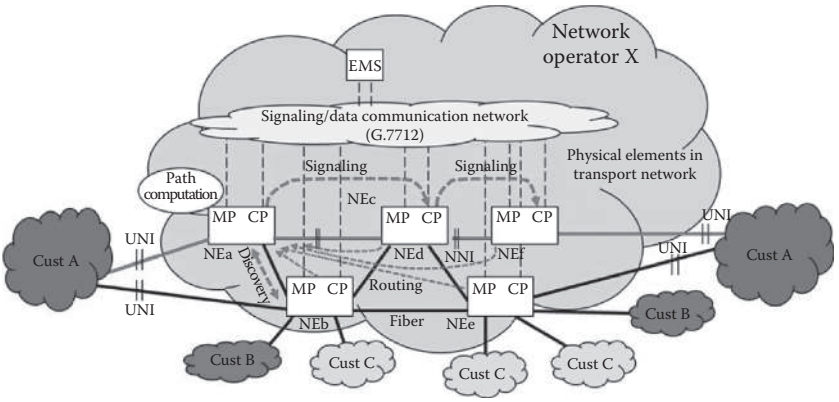
Neighbor discovery is the communication of identity between neighboring NEs. It is enabled by exchanging addresses over the control channel, allowing each device to build up an accurate inventory of link identities and remote link endpoints, and detecting misconnections. Discovery requires the use of an in-band control link or, if an out-of-band control link is used, an additional identifier carried in the OH or the wavelength itself.

The control plane routing protocol is then used to disseminate local link topology and link use to all network elements within the control domain so that each NE builds a complete topology map. This makes it possible for the management system to retrieve the full network topology and status by contacting a single NE. Optical network routing differs from IP routing in that routing is needed only when a connection is initially provisioned, not for every packet. This reduces the criticality of the routing protocol because data will flow correctly on existing connections even if the routing protocol is disrupted.

The topology database is used by the source node of a new connection to compute the optimized path for the connection. Distributed path computation reduces the load on the management system and allows the network to recover rapidly from failures by having recovery paths computed by the affected source nodes.

Finally, new services can be provisioned using a signaling protocol between each of the NEs in the connection path, reducing the requirements on the management system and drastically reducing the latency of connection setup and connection restoration after failure. Control plane signaling protocols set up the connection along a precomputed path end to end, using an explicit route object inserted into the setup message at the source node. As shown in Figure 15.2, the management system plays a reduced role for offline device management, whereas the NEs communicate in real time using signaling to set up the end-to-end path.

Sophisticated planning tools can retrieve the instantaneous topology, status, and occupancy of network links and nodes from an active



CP, control plane; Cust, customer; DP, data plane; EMS, element management system; MP, management plane; NE, network element; NN, node interface; UNI, user-to-network interface.

Figure 15.2 Control plane.

network element and perform an analysis of network use. If a more efficient or better performing configuration of the network is computed, the control plane will automatically adjust its routing of new connections once the new configuration has been installed.

How Can SDN/OpenFlow Improve Optical Network Control?

Goals of Applying SDN/OpenFlow to Optical Networks

Although the optical control plane has been successfully deployed in many carrier networks and has reduced the capex and opex of these networks, there are major potential benefits to adopting SDN/OpenFlow for optical network control, as follows:

- A programmatic, abstracted interface for application awareness and greater influence over the network. In the current optical network, the applications and the network are independent, missing potential efficiencies from the coordination of demand and resources. In control plane models such as generalized multiprotocol label switching (GMPLS), the network is expected to react independently to requests from client systems distributed at the network edge, making global coordination of actions more difficult.
- An improved service development and deployment. Because service processing is distributed across network nodes, deployment

of a new service in a distributed control environment may involve upgrading the software at each node in the network. Not only does this introduce deployment timing problems, but it also requires software development coordination across platforms and extensive testing to avoid potential interactions with embedded software and services. The use of OpenFlow/SDN to separate software and hardware would allow service software to be developed on server platforms rather than embedded systems.

- Multiple levels of abstraction. Introduction of the OpenFlow/SDN technology support allow optical network virtualization, presenting a different view of the network depending on application requirements. Different applications may need different levels of information and control over connectivity to specified destinations. The Quantum application programming interface (API), for example, in the OpenStack software suite allows an application to simply request a new layer 2 network connecting peer machine [11]; a similar abstraction would allow the introduction of very high bandwidth point-to-point services with greater customer controllability.
- Cost reduction. Potentially, greater separation of software and hardware using SDN/OpenFlow may reduce the cost of optical network equipment by reducing the processing requirements and software development costs for the network element and centralizing software on common platforms. How great the cost reduction would be is unclear, however, because the cost of optical equipment is primarily in the hardware, the photonics, and the associated electronic components, rather than in the software.

Potential Applications for SDN/OpenFlow in Optical Networks

Packet–Optical Interconnection (POI)

An initial application for SDN/OpenFlow is for the control of multilayer packet/optical networks. Packet-optical transport systems (POTS) incorporate both packet switching and optical switching/transmission in one device, simplifying the aggregation of packet transport into optical pipes and allowing for the efficient grooming

of traffic into the carrier's optical network [12]. The use of SDN/OpenFlow for control is relatively easy here because OpenFlow already contains the control semantics for directing the mapping of packet traffic into virtual ports (defined in the OpenFlow 1.2 specification [13]), and these virtual ports can be mapped by a management system or an optical control plane into point-to-point optical paths connecting the POTS systems across the wide area network (WAN). The use of OpenFlow allows the service provider to specify more flexible mapping based on the different components of the packet header and flexible encapsulation and decapsulation actions depending on the desired service, whereas the optical paths provide high bandwidth connectivity with guaranteed bandwidth and performance between sites. This could allow the carrier to introduce new services based on access control or selective class of service.

Data Center Interconnection and Network Virtualization

The OpenFlow/SDN technology is already seeing its widest deployment within the data center, where topology and traffic patterns can be controlled and equipment tends to be homogeneous [14]. A logical extension is to apply OpenFlow/SDN to the data center interconnection across the WAN. Inter-DC traffic comprises database distribution and synchronization between geographic sites, virtual machine (VM) migration, and transfer. Much of this traffic will be generated by the need for geographic distribution to meet backup requirements and movement of workload from one cloud to another. Flexibility in workload placement and movement among the pool of provider DCs contributes to a reduction in the total DC resources through the expansion of virtualized asset pools. This will create an important potential source of economies for SPs (Figure 15.3).

The cloud backbone network interconnecting provider DCs poses challenges that the OpenFlow-based SDN is ideally suited to address.

User self-service paradigms, application operational time variations, and a significantly transactional component to many of the traffic types all create an intrinsically and significantly time-variable character to the traffic among DC connection points on the backbone network. Survivability and recovery from disasters or major outages

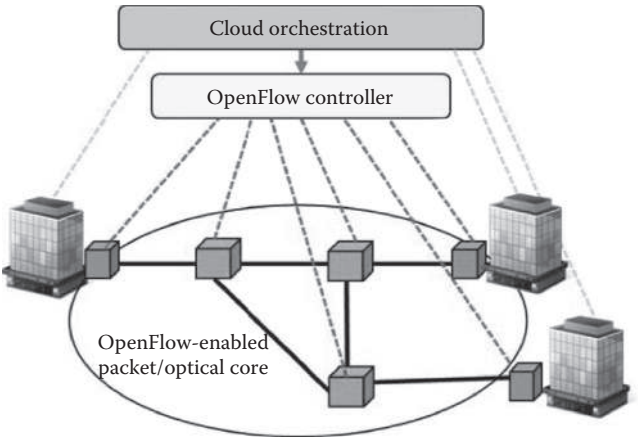


Figure 15.3 Data center interconnection application.

may add large volumes of unpredictable, transactional traffic on the inter-DC backbone.

Under existing paradigms, both shared packet networks and dedicated connection require adequate bandwidth capacities to support peak traffic loads on all paths and through all aggregation and switching points. This implies a design-for-worst-case planning paradigm, effectively requiring an overdesign to add slack capacity (i.e., peak vs. average or lowest required capacity) to the network.

Under a centralized network control layer that maintains a global view of network resources and controls their allocation in response to evolving traffic demands—the SDN paradigm—the various trends and challenges previously described may be addressed. The cloud orchestrator and network control layer are global; both the demands and the use of network resources may be globally optimized. For example, the network control layer may see that concurrent major data transfers can be accommodated by sending each over a different network path, despite node or link saturation along default routes.

Private Optical Networks

A special case application of SDN/OpenFlow for data center interconnection is for smaller private optical networks that a data center operator may deploy to connect their locations. If these networks cover a relatively small geographic area, the simplicity of a centrally

run SDN/OpenFlow control plane designed for a small private network may be very attractive to the data center operator, where their requirements for scalability, reliability, etc., are not as extensive as would be for a service provider. This application could be the driver of early, limited deployments of OpenFlow/SDN for optical networks.

Extending OpenFlow/SDN to Optical Networks

What Are the Challenges to OpenFlow/SDN for Optical Networks?

Although significant research has been done on the extension of OpenFlow/SDN to optical networks, there are still some challenges that will need to be overcome before there is real implementation and deployment in service provider networks.

OpenFlow Extensions for Circuits OpenFlow (e.g., OpenFlow v.1.0 and v.1.3) supports the configuration of matching of incoming packets based on port, Ethernet header, IP header, and transmission control protocol (TCP) port values and forwarding rules to modify headers and selectively forward packets to outgoing ports. Basic extensions for circuit control have not yet been integrated into the OpenFlow specification, although research and prototyping has been done at both layers 1 and 0, as will be described below. The complexity of these extensions depends on the level of control, especially on the level of functionality, as follows:

1. The mapping of input to output timeslot and/or wavelength
2. The control of optical transceiver characteristics such as modulation type, power levels, dispersion compensation, etc.
3. The control of the internal functions of the switch, such as adaptation between different TDM layers at layer 1 or wavelength conversion at layer 0

OpenFlow uses a simple Match/Action Table model of the switch that does not easily model internal characteristics, such as port-to-port wavelength connectivity limitations in ROADMs, and assumes that much information about switch constraints is preconfigured in the controller. Similarly, the model focuses on the actions within the switch and not the links between switches, whereas much of the complexity of transport deals with link characteristics.

Optical Impairments At the photonic layer, the handling of optical impairments and characteristics will be critical. This can be divided into two main aspects:

1. Path computation and set up of cross-connection at intermediate nodes. Path computation would be done in the controller and would need to consider the impact of optical impairments on end-to-end signal-to-noise ratio; cross-connection control would need to specify the matching of incoming and outgoing ports, assuming that wavelength continuity is preserved across the switch. If wavelength conversion is possible, then incoming and outgoing wavelength or waveband must also be controlled.
2. Setting of transmitter and receiver for compatibility with each other and matching with the optical end-to-end path. Within a frequency range that has been cleared end to end, there may be a variability of the characteristics of the signal sent by the transmitter and detected at the receiver, such as modulation type, power level, FEC coding, etc., which must be set by the controller; this setting may be done using individual parameters or by groupings of parameter settings that have been standardized for interoperability, such as the application codes standardized in the ITU-T Recommendations for this purpose [15].

Lack of Visibility into the Data Stream In transport networks, the objective is to convey information transparently between endpoints. The information is defined by characteristics such as port, timeslot, and wavelength defining the data stream rather than information carried within the data stream such as packet header fields. As a result, the controller cannot request actions based on an analysis of the data stream, but only actions based on the port, timeslot, and wavelength defining the data stream. This limits the functionality of the controller interface compared with packet networks.

Technology Specific Discovery Discovery in OTNs cannot be done using the Packet_In/Packet_Out method used for packet network discovery because there is no visibility into the data stream. Instead,

the controller must be able to take advantage of technology-specific discovery mechanisms such as setting of the trail trace or other header information or, alternatively, exchange of discovery information via the DCC/GCC/OSC control channels of the optical path.

Service Provider Policy and Interoperability Service provider networks are typically made up of diverse equipment from multiple vendors and are used to provide service to many different customers. Deployment of SDN/OpenFlow in the service provider network will need to be able to support such an environment by:

- Supporting multiple domains of differing equipment types and vendors, where a single controller instance may not be sufficient for control purposes. Multiple controllers will need to be coordinated in a multidomain network, but there is no standard controller-to-controller interface. Another approach to coordination may be the use of hierarchy where a parent controller coordinates the actions of multiple child controllers; however, this will introduce further requirements for controller-to-controller interoperability and testing.
- Supporting service provider needs for injecting policy and authorization over the control of network resources. OpenFlow/SDN already has the concept of a FlowVisor [16]. A FlowVisor is a mechanism for partitioning control so that a particular client controller only sees and controls a subset of a controlled network; however, the policy aspects of configuring what resources are controlled by which client will need further specification. In general, security is an area of OpenFlow/SDN that is still a work in progress.

Reliability and Scalability Service provider networks cover large geographic areas where both the number of network elements and the geographic distance can be a challenge for SDN/OpenFlow control structures. At the speed of light in fiber (~300,000 km/s), control messages require an approximately 30-ms round trip to cross 5000 km, a significant amount of time relative to 50-ms standards for transport network actions such as protection switching in case of failure.

Some functions will clearly need to be controlled locally to meet standard time frames for recovery actions. The number of network elements and the requirements for very high availability of service provider networks (aiming at 0.99999 availability) will require that controllers have active backups with state synchronization and fast failover time, features that are still under development for controller implementations.

Research on OpenFlow/SDN for Optical Networks

Ciena/Stanford Prototyping As a proof of concept, Ciena and Stanford University cooperated in 2009 on a prototype OpenFlow-enabled packet and circuit network, using the Ciena CoreDirector (CD) CI SONET/SDH switch and a Stanford-developed OpenFlow controller/application that set up, modified, and tore down L1/L2 flows on demand and dynamically responded to network congestion. The network and application was demonstrated at the SuperComputing 2009 Conference as the first implementation of integrated packet and circuit control based on OpenFlow [17] (Figure 15.4).

At the start of the demonstration, connectivity between the CD switches and the OpenFlow controller is established over an out-of-band Ethernet control network. The controller was initially configured

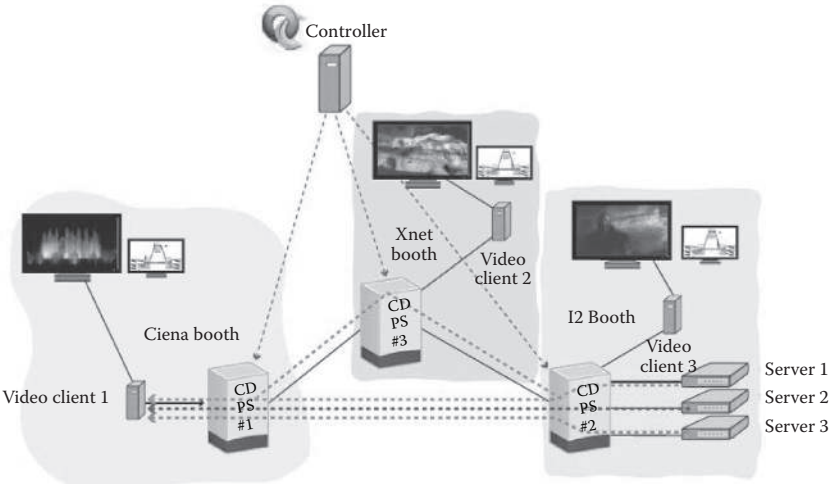


Figure 15.4 Ciena/Stanford prototype demonstration.

with the identities of the switches and used communication over the OpenFlow interface to build a switch/topology database. The controller then preprovisioned a SONET/SDH Virtual Concatenation Group (VCG) between the CDs, capable of carrying Ethernet private line connections. After this initial startup phase, a video client would make a request for a video from a remote streaming video server. The request is redirected to the OpenFlow controller using the Packet_In command, and the controller responds by directing CDs 1 and 2 to create an internal VLAN corresponding to the client port (in CD 1) and the video server port (in CD 2), and map the VLAN into the VCG virtual ports, thereby enabling the packet flow to be transported over the VCG. All subsequent packets (in both directions) for this client-server pair are matched at the CD Ethernet port using the existing flow definitions and are directly forwarded in hardware. At the client side, the packets received from the VCG are switched to the client port based on the VLAN tag, which is then stripped off before the packets are forwarded to the client PCs, where the video is displayed on the screen. Using OpenFlow, it was possible to display both circuit and packet states in real time.

The extensions required for OpenFlow included the following:

- OpenFlow specifications (especially OpenFlow 1.0 [18], which was the basis for the prototype) already support control over matching and actions for the input and output ports and packet header information fields below:

Physical	–Input port
Ethernet	–VLAN ID
	–Source address
	–Destination address
	–EtherType
IP	–Source address
	–Destination address
	–Protocol number
TCP	–Source port
	–Destination port

- In addition to the support of the input port and the L2 to L4 packet header information for matching of incoming packets and configuration of these on egress, extensions were made

to allow L1 circuit parameters to be matched and configured and the creation of VCG that, in SONET/SDH, can be hitlessly enlarged or reduced in bandwidth depending on the following traffic requirements:

Physical	–Input port/fiber
Wavelength	–Input wavelength
Electronic group	–VCG ID
Electronic TDM	–Starting timeslot –Signal type

- Lastly, extensions were made to pass L1 topology information from the switch to the controller (especially discovered peer's switch and port IDs), allowing the use of technology-specific discovery mechanisms in the switch because the Packet_Out function of OpenFlow could not be used over an L1 interface.

The extensions made to the OpenFlow protocol were documented in a publically available software package called pac.c [19] and have been used in subsequent research projects on OpenFlow-based circuit control.

OpenFlow in Europe—Linking Infrastructure and Applications (OFELIA) Project Another major research project on the application of OpenFlow/SDN to optical networks is the OFELIA project. The goal of the OFELIA project is to create an experimental facility that allows researchers to control the network that they are using for carrying experimental data precisely and dynamically using OpenFlow-based control. The OFELIA work focuses on the virtualization of the optical network using standard interfaces, including both OpenFlow and GMPLS.

OFELIA is a large research effort, with an European Commission (EC) budget of €4,450,000, a 3-year life span (2010–2013), and a network of five federated island domains, including networks in the United Kingdom, Switzerland, Italy, and Germany [20].

One of the initial studies by the University of Essex [21] looked at the pairing of OpenFlow and GMPLS through the OpenFlow control of the packet mapping at access points and the use of the GMPLS

user network interface (UNI) user-network interface to request paths across the optical network, as shown in Figure 15.5.

In this case, the OpenFlow protocol itself is unchanged, and another interface (GMPLS UNI) is used for the control of the optical network. In a further experiment, extensions were defined to the OpenFlow interface that built on the pac.c work, adding flexible labels rather than strict SONET/SDH timeslots and control of adaptation as well. In this model, the OpenFlow protocol acts on a virtual header with circuit characteristics [22].

CCT ID	in port	out port	label in (e.g. encoding, ST, G-PID)	label out (e.g. encoding, ST, G-PID)	adaptation actions

This format allows great flexibility because the label can correspond to the timeslot in the electronic domain or the wavelength in the optical domain; furthermore, it is possible to specify the type of adaptation to be used between layers. Further studies of the full integration of the OpenFlow control of packet and optical versus combinations of OpenFlow and distributed optical control suggest that the added flexibility of full integration may come with some additional cost when it involves the redesigning of the optical

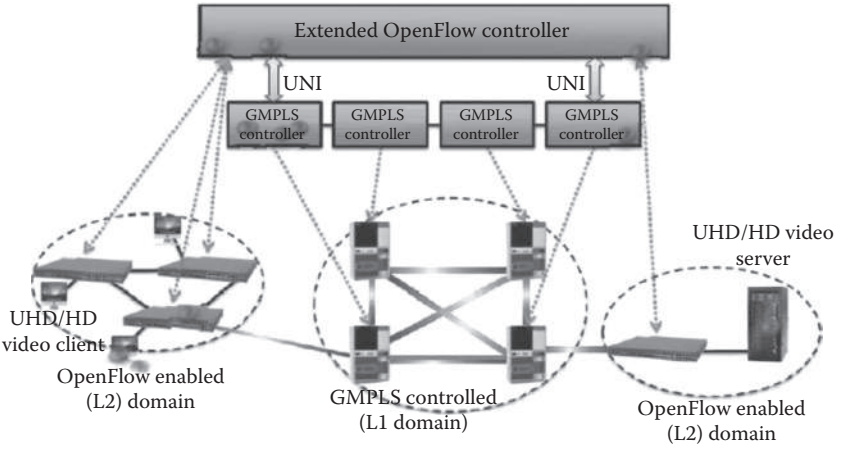


Figure 15.5 OFELIA study using GMPLS UNI (UHD/HD, ultra high definition/high definition).

control plane, and may not be the least costly solution, although architecturally consistent.

How Will OpenFlow/SDN Be Deployed in Carrier Optical Networks?

OpenFlow/SDN Overlay with Transport Tunnels for POI

As discussed above, the OpenFlow interface already supports the capacity for the control of packet forwarding at L2 and above and the concept of virtualized ports for the ingress and egress that can be the endpoint of a configured packet or circuit tunnel. This can be applied easily to packet/optical interconnection, where OpenFlow is used to control packet mapping into optical circuits, and the optical circuits are controlled separately through a management system, a distributed control plane, or other methods such as stateful path computation element (PCE) (described in more detail under Standards). In this deployment, OpenFlow would be used for one (packet) layer to specify the matching of incoming packets and forwarding them into a virtual port. OpenFlow is only needed for the subset of network elements supporting packet interfaces for customer edge services, whereas the core of the network (optical paths) are set up and controlled by an internal mechanism. Traffic through the tunnels passes through intermediate switches without visibility to the controller, and traffic engineering through the core is done independently.

Recent modeling of the POI control through OpenFlow has taken a different direction than the initial research, which focused on the treatment of wavelength, timeslot, etc., as additional match fields. Instead, the current modeling uses the concept of logical ports introduced into more recent versions of OpenFlow. In this model, the physical port on the switch may have multiple associated logical ports, and each logical port may have characteristics, such as wavelength or timeslot, which are modifiable by the controller (alternatively, the model may include a separate logical port for each wavelength or timeslot and use the match table to configure the mapping of a packet flow to a particular logical port). A generalized model for port characteristics in transport networks would include bit error rate, alarms, and other information available from the digital framing and, possibly, associate link characteristics with either the logical port or the associated physical port (Figure 15.6).

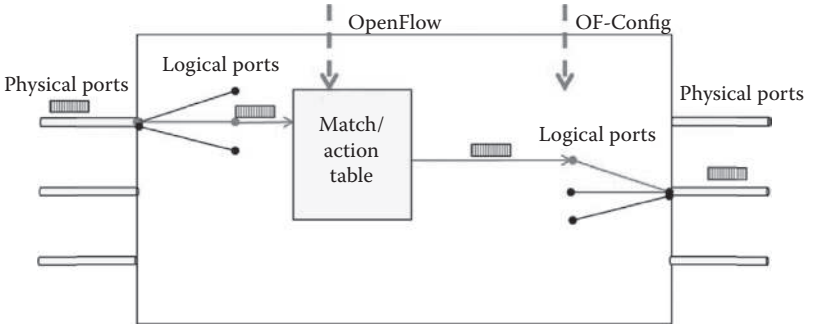


Figure 15.6 OpenFlow POI model.

OpenFlow Overlay Over the Control Plane (with Abstraction)

With extensions for the control of circuit mapping and packet forwarding, analogous to what was explored in pac.c and OFELIA, OpenFlow can be used directly for control over optical switching elements. To speed up the deployment of such capability in a carrier environment, which typically consists of multiple domains of deployed equipment, it would be more cost effective to deploy OpenFlow/SDN as an overlay rather than introduce an OpenFlow/SDN agent on every network element and add connectivity from each network element to a controller. Moreover, scalability and reliability requirements would be greater. Instead, an overlay deployment of OpenFlow/SDN would use OpenFlow interfaces only to an element management system (EMS) or a subset of network elements, where OpenFlow can be used by the application as a programmatic interface for the control of paths across the network, but the actual instantiation and direct control of resources is managed separately.

The intermediate system would mediate between the requests sent by the client controller and the actual control of network devices, providing an abstract model to the client controller that appears as a real network that is dedicated to that controller, where in fact the real network is being shared by multiple client controllers. This simplifies the function of the individual controller and provides policy control over the resources that are allocated to each application. Performance issues may be a concern here because the mediation function adds latency to control traffic going both from the client controller to the device and from the device to the client controller (e.g., event notifications) (Figure 15.7).

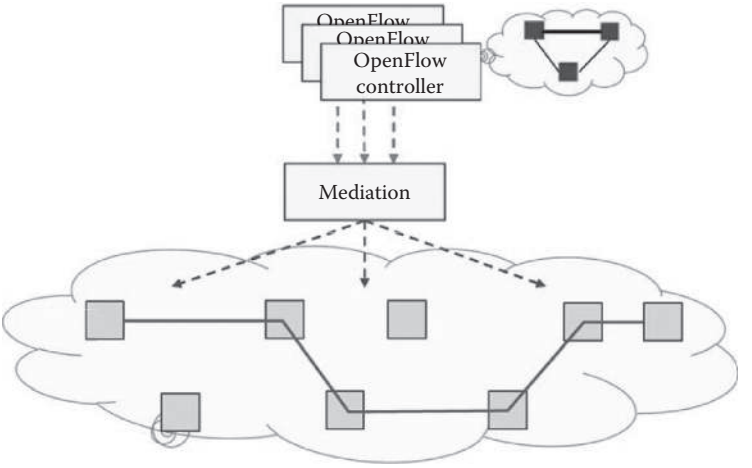


Figure 15.7 OpenFlow overlay with abstraction.

Direct OpenFlow Control of Packet/Optical

Although the overlay deployment addresses the programmability of optical networks and the improved separation of service and equipment software for faster service deployment and innovation, it does not affect the cost of network equipment, as overlay implies that the OpenFlow/SDN is deployed as an extra interface in addition to whatever legacy control structure is in place. If the goal is to achieve savings by the complete separation of software and hardware, this entails a fully centralized OpenFlow/SDN structure where the software on the network equipment is limited to an agent and to whatever is necessary to control local actions (e.g., protection) and element management. This is also the greatest change from the existing method of network operation, making it the most difficult alternative to implement, test, deploy, and manage for the service provider.

In a fully centralized OpenFlow/SDN control architecture, every network element has an interface to the controller, and the controller impacts all switching actions, including multiple layers if the network element supports multiple layers of switching technology. The controller must then be aware of all supported and unsupported actions, including any limitations on adaptation from one layer to another or connectivity between an incoming port and an outgoing port. To allow data to flow across the network, the controller must talk to each

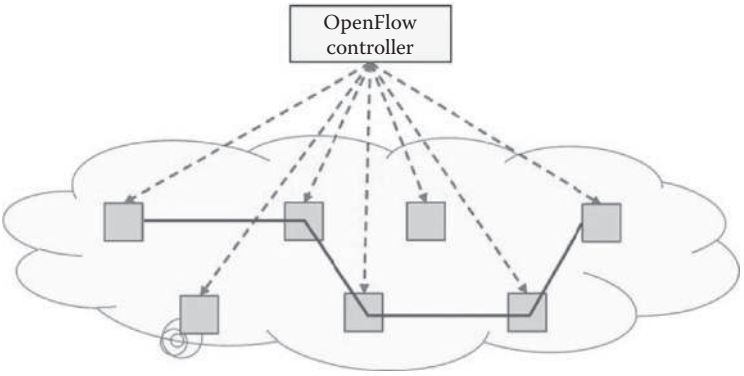


Figure 15.8 OpenFlow direct/centralized control.

network element along the datapath and configure the matching and forwarding actions that allows data to flow from the input port to the desired output port connecting to the downstream node (Figure 15.8).

The issue with a fully centralized control is as much related to the implementation and deployment as to the technology: carriers will need to verify that the solution is scalable and reliable, and even then, deployment will be gated by having to modify existing operations systems and procedures.

Standards and Interoperability

Going forward, there are several activities that have been established in the industry to develop common standards for the OpenFlow/SDN application to optical networks. The main activities are in the Open Networking Foundation (ONF) and the Internet Engineering Task Force (IETF).

Open Networking Foundation ONF is a nonprofit consortium dedicated to the development and standardization of SDN interfaces, particularly, the OpenFlow protocol and the OpenFlow Configuration and Management (OF-Config) protocol. The mission of OFN is to commercialize and promote SDN and the underlying technologies as a disruptive approach to networking. Its activities are directed by a board made up of members of the user and service provider

community. The ONF, by virtue of its control over OpenFlow, is the primary body for the development of SDN standards [23].

ONF has two groups, in particular, that are relevant to the creation of SDN for transport networks: the Extensibility Working Group (WG) and the New Transport Discussion Group. The Extensibility WG of ONF develops the detailed extensions to the OpenFlow specification that are deemed necessary to improve or extend the functionality of OpenFlow. Any technical extensions to the OpenFlow protocol must be passed by the Extensibility WG.

The New Transport Discussion Group was established in 2012 within ONF as a forum for discussing the application of SDN/OpenFlow to transport networks, including both optical and wireless networks in its original scope. The group is currently focusing on optical networks, with the objective of defining the use cases for SDN/OpenFlow over optical networks, the requirements for service provider and private network applications, and the recommendations for extensions to the OpenFlow protocols.

IETF Standards Early efforts to incorporate SDN concepts into Internet standards led to the scheduling of IETF Birds of Feather (BoF) sessions on software-driven networking [24] and cross-stratum optimization (CSO) [25]. The former focused on the control plane architecture that would incorporate a centralized controller that interfaced with network elements to control connectivity at an abstract level. The latter focused on the potential use cases and benefits that could result from the coordination of applications and the optical network, the strata of its name. Both efforts were eventually judged to be not sufficiently of interest to IETF and were terminated with no follow-up standards effort.

Efforts in IETF have now focused on a project called Interface to Routing System (I2RS) [26] that takes a different direction that is less disruptive to the current routing environment. This approach focuses on the creation of a new programmatic interface into the router, which will support greater visibility into the routing information base and greater ability for the application to control forwarding decisions. Although, in concept, this will address the goal of programmability, it does not address service deployment or cost-reduction goals.

PCE and Stateful PCE Because of the limitations of existing distributed routing mechanisms, work on the use of PCEs [27] for complex path computation began in 2006. PCE separates the path computation function from the other functions in a network element, allowing path computation to be migrated to a centralized server. PCE is a broadly applicable technology, which can potentially solve the problems with multidomain routing in optical networks and help with problems of complex path computation for all-optical networks.

The basic entities in a PCE system are the path computation client (PCC) and the path computation entity (PCE). The PCC uses the PCE protocol (PCEP) [28] to request a path from a source to a destination, along with other information, such as transport requirements, constraints, and the type of optimization. The PCE can be implemented on a network element or on a dedicated server and does path computation with its local database.

What ties PCE with SDN/OpenFlow is the concept of stateful PCE [29], where the PCE not only computes the path for a connection, but also controls the state of that connection, including causing the connection to be modified, rerouted, or terminated. Recently, it has been proposed that the PCE be able to initiate new connections as well, in which case, it takes on the functions of a central controller as in the OpenFlow, with the exceptions that:

- Its control of matching and mapping actions is limited to the context of a single layer connection (cannot operate over a flexible set of header values as in OpenFlow); and
- It relies on a distributed signaling protocol to carry out the actions it specifies, for example, to modify a connection, it directs the source node to send out the necessary signaling messages rather than interact with each network element in the path.

In Figure 15.9, the PCE provides an API for communication with client applications and has network topology stored internally. When requested, it both computes the desired network path and initiates the connect setup process at the source NE.

A variety of potential extensions for PCE-driven network action, coordinated by applications, has been captured in a recent IETF draft in this area [30].

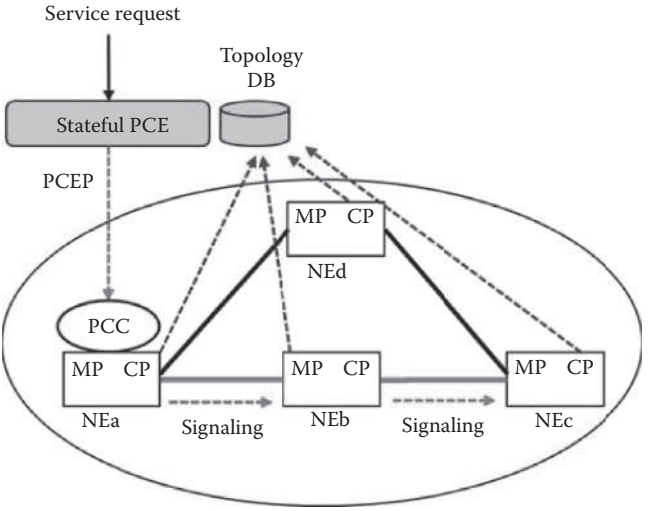


Figure 15.9 PCE with stateful PCE. DB, database; MP, management plane; CP, control plane; PCC, path computation client.

Optical Internetworking Forum (OIF) One other group that may be involved is OIF. OIF is a group of carriers, system vendors, and component vendors that focuses on interoperability and agreements for the deployment of optical networks. OIF is led by a strong Carrier WG that provides carrier requirements for optical technology, both at the system and component levels, and includes major service providers such as AT&T, Verizon, Deutsche Telekom, France Telecom, China Telecom, and NTT. An example of OIF's work on requirements is the work that was done on the definition of a framework for service provider long haul 100-G transmission [31]. For SDN/OpenFlow, OIF has initiated work to identify use cases and carrier requirements for transport SDN through its Carrier WG. OIF could play a significant role in establishing the requirements for carrier-grade OpenFlow/SDN and driving deployment into service provider networks.

Conclusions for the Future

OpenFlow/SDN will need several modifications to be suitable for optical network control, both extensions to the protocol itself to control circuits (with no data stream visibility) rather than packets and progress in implementations to support service provider scalability,

security, and reliability requirements. Initially, packet-oriented OpenFlow/SDN can be used in overlay deployment to control packet mapping and combined with existing methods of optical path provisioning to offer improved packet/optical interconnection services.

An extensive body of research and prototyping does exist for the application of OpenFlow/SDN to optical networks at layers 1 and 0, establishing basic functional requirements that would need to be addressed for circuit control. These may be the basis for limited scale deployment in private optical networks, where the requirements on implementation are less stringent. Deployment in the overlay mode where the OpenFlow/SDN interface serves as a service interface and is mediated by a transport network FlowVisor can be an early method of deploying capabilities in a service provider environment to support service innovation.

Eventually, the use of direct OpenFlow/SDN control of optical networks may emerge as a control architecture within domains of optical equipment as high availability controller implementations become available. Further work on the interconnection of controllers is needed for OpenFlow/SDN to scale to carrier environments that typically consist of multiple domains. How rapidly such deployment occurs depends on the ability of implementers to develop hardened, scalable versions of OpenFlow/SDN controllers and how real the advantages will be for cost and service development and deployment.

16

Security Issues in SDN/OpenFlow

Contents

Introduction	415
SDN Security Concerns	416
Enabling Fast Failure Recovery in OF Networks	418
Network Intrusion Detection and Countermeasure Selection (NICE) in Virtual Network Systems	420
FRESCO: Modular Composable Security Services for SDNs	423
Revisiting Traffic Anomaly Detection Using SDN	425
Language-Based Security for SDNs	427
Scalable Fault Management for OF	429
A Dynamic Algorithm for Loop Detection in SDNs	432
Discussion	433
Conclusion	433
References	434

Introduction

Software-defined networking (SDN) is a new approach to networking. It was invented by Nicira Networks based on their earlier work at Stanford University, University of California at Berkeley, Princeton University, and CMU. The goal of SDN is to provide an open, user-controlled management of the forwarding hardware in a network. SDN exploits the ability to split the data plane from the control plane in routers and switches. The control plane is open and controlled centrally with SDN while having the commands and logic sent back down to the data planes of the hardware (routers or switches). This paradigm provides a view of the entire network and helps make changes centrally without a device-centric configuration on each hardware. The OpenFlow (OF) standard and other open protocols help manage the

control planes and allow for precise changes to networks or devices. SDN works by creating virtual networks that are independent of physical networks. To achieve this, it separates the control plane from the data plane and allows the user to control the flow of traffic in the network. Figure 16.1 illustrates the difference between the traditional network and the OF-based SDN.

An OF system typically includes the following three important components:

1. The OF switch. OF provides an open protocol to program the flow table in different switches and routers. An OF switch consists of at least three parts: (1) a flow table, with an action associated with each flow entry; (2) a channel, allowing commands and packets to be sent between a controller and the switch; and (3) the OF protocol, which provides an open and standard controller to communicate with a switch.
2. Controllers. A controller adds and removes flow entries from the flow table on behalf of experiments. A static controller might be a simple application running on a PC to statically establish flows to interconnect a set of test computers for the duration of an experiment.
3. Flow entries. Each flow entry has a simple action associated with it; the three basic ones (that all dedicated OF switches must support) are (1) to forward this flow's packets to a given port, (2) to encapsulate and forward this flow's packets to a controller, and (3) to drop this flow's packets.

SDN Security Concerns

Before we discuss the security issues of SDN, let us quickly review the traditional networks' vulnerability and protection as summarized in Table 16.1.

SDN creates some new targets for potential security attacks, such as the SDN controller and the virtual infrastructure. Besides all the traditional network attack targets, SDN has new target points, as follows:

- SDN controller: traditional application, server, and network attacks

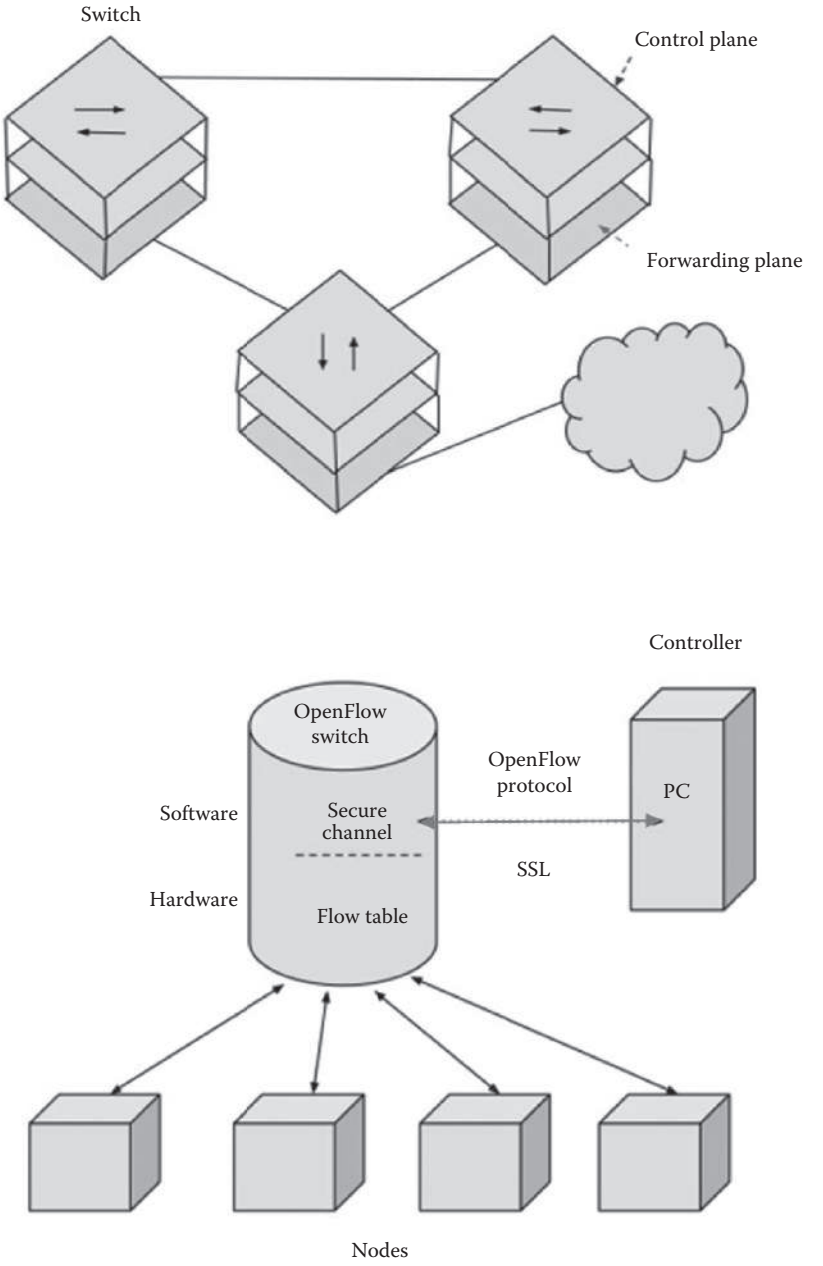


Figure 16.1 Traditional network architecture (top) and OF architecture (bottom) (SSL, secure socket layer).

Table 16.1 Traditional Network Vulnerability and Protection

EXAMPLE ATTACK TARGET LAYER	EXAMPLE ATTACK POINTS	VULNERABILITY EXAMPLES	PROTECTION EXAMPLES
Applications	Network apps, general apps (e.g., database)	– General: cross-site scripting, buffer overflow, SQL injection – Net: DNS cache poisoning	– General: firewall, intrusion, detection, intrusion, prevention – Net: DNSSec, SSL
Servers	Transport, OS, hypervisor	– TCP: SYN flood, SYN/ACK scan, spoofed RST, hijack – UDP: smurf DoS attack, spoofed ping-pong	– Encrypt session: SSH, IPSec, intrusion detection/prevention
Network	Routers, switches, virtual switches	– IP/routing: MIM routing attack, FIRP attack, IP spoofing, ping flood, ICMP destination unreachable, smurf attack, source attack – FC: target/initiator spoofing – MAC (FCF) spoofing – ARP cache poisoning – Physical link tap	– IP/routing: IP ACL filters, firewall, intrusion detection, OSPF with IPSec, split horizon, infinite hop count detection, override source routing – FC: zoning, FC ACL filters – Ingress/egress MAC ACL filters, VLANs – Physical security – Authentication protocol

- Virtual infrastructure: traditional application and server attacks on the hypervisor, virtual switch, and virtual machine (VM)
- Network: OF protocol for OF-enabled devices

In the following section, we will describe some important OF/SDN security schemes.

Enabling Fast Failure Recovery in OF Networks

It is important to have some robust restoration options available in OF networks. In Ref. [1], the addition of a fast restoration mechanism for OF networks is proposed, and its performance is evaluated by comparing the switchover times and the packet loss to the existing restoration options in a current OF implementation. It discusses some mechanisms implemented at the OF and Nox software to recover from a link failure. It also discusses the limitations of these mechanisms in enabling fast recovery in OF networks.

In the case of OF mechanisms, the failure can be recovered if a new correct flow entry is added in OF switches after the failure occurs. The recovery from the failure depends on the time when the OF switch requests the flow entry from the controller. Hence, the recovery from failure depends on the life of the flow entries in the flow table. The life of flow entries is associated with two intervals: idle timeout and hard timeout. Idle timeout indicates the time when the flow entries should be removed because of the lack of activity. It is the time interval of a flow entry with which the OF switch has not received the packet of a particular flow of that entry. Hard timeout implies the maximum life of flow entries, regardless of activity. OF switches remove their flow entries from the flow tables after the expiration of one or both intervals. Recovery from failure is directly proportional to the value of the aforementioned intervals.

In the case of Nox mechanisms, the recovery from failures is possible with a new flow entry only if the controller also knows about the failure. Otherwise, the controller may add an incorrect entry in the OF switches. Thus, recovery depends not only on hard and idle timeouts but also on mechanisms running in the network to detect the failure. Nox implements L2-Learning and routing mechanisms to recover from a failure. They implemented L2-Learning in C++ and Python. The former is called the L2-Learning switch, and the latter is called the L2-Learning pyswitch. They behave differently to recover from a failure.

Fast recovery is only possible if the incorrect flow entries are flushed from all the OF switches and new entries are installed immediately after detecting the link failure in the existing path. This will be possible with the help of a controller, only if the controller (1) knows about the failure, (2) remembers the older path that was established by adding flow entries in OF switches, (3) is able to calculate a new path, and (4) knows all the current flows in the network.

In fast restoration, the controller performs a set of actions to restore the path between affected hosts. The initial task is to check whether its calculated older paths among end hosts are affected or not. If these are affected, then the controller calculates the new path for those end hosts. Other than this, the controller also checks if it has added flow entries in OF switches regarding the older faulty path. If yes, then it deletes the flow entries from all the OF switches

regarding the older path and adds flow entries in all the OF switches for the new path.

In Ref. [1], they used Ubuntu v.9.04 for the installation of OpenVSwitch v.1.1.0 and Nox v.0.9.0. More than 11,000 ping packets were sent from Client 0 to Server 1 with an interval of 10 ms. Then, the number of ping replies received by Client 0 was calculated. Hard and idle timeouts for flow entries are set to 20 and 10 s, respectively. The network routing loops can be easily removed by using any loop-free mechanism, for example, by building a spanning tree in the topology. The failure recovery time was investigated in the OF network when the outgoing path is affected by link failure. For fast restoration, we need an efficient scheme to calculate paths. A predetermined mechanism was used in Ref. [1] to calculate the path. The OF architecture allows the implementation of restoration options that are much faster than medium access control (MAC) reconvergence (routing and L2-Learning pyswitch) or the client-initiated recovery with an address resolution protocol (ARP) request (L2-Learning switch). Their fast restoration mechanism can be integrated in any mechanism where a controller is able to detect the failure by some means. In their fast restoration mechanism, the flow is able to switch to another path within 12-ms interval regardless of the time left before the expiration of the flow entries.

Network Intrusion Detection and Countermeasure Selection (NICE) in Virtual Network Systems

In Ref. [2], NICE in virtual network systems was investigated. The Cloud Security Alliance (CSA) survey shows that, among all security issues, the abusive use of cloud computing is considered as the top security threat. Attackers can exploit vulnerabilities in clouds and use cloud system resources to deploy attacks. The convention schemes that patch known security holes in cloud data centers, where cloud users usually have the privilege to control software installed on their managed VMs, may not work effectively and can violate the service-level agreement (SLA). In Ref. [2], the authors propose NICE to establish a defense-in-depth intrusion detection framework. For better attack detection, NICE incorporates attack graph analytical procedures into the intrusion detection processes. In general, NICE includes two main phases:

1. They deploy a lightweight mirroring-based network intrusion detection agent (NICE-A) on each cloud server to capture and analyze cloud traffic. NICE-A periodically scans the virtual system vulnerabilities within a cloud server to establish a scenario attack graph (SAG). NICE's decision about whether to put a VM in the network inspection state will be based on the severity of the identified vulnerability toward the collaborative attack goals.
2. Deep packet inspection (DPI) is applied when a VM enters the inspection state and/or when virtual network reconfigurations can be deployed to the inspecting VM to make the potential attack behaviors prominent. In Ref. [2], they talk in detail about how to use attack graphs to model security threats and vulnerabilities in a virtual network system and propose a VM protection model based on virtual network reconfiguration approaches to prevent VMs from being exploited.

Different types of models are explained in Ref. [2]: (1) the threat/attack model: It is assumed that an attacker can either be outside or inside of the virtual networking system, and his primary goal is to exploit vulnerable VMs and compromise them as zombies. To improve the resiliency to zombie explorations, the protection model focuses on virtual-network–based attack detection and reconfiguration solutions; (2) the attack graph model: This is a modeling tool that illustrates all possible multistage, multihost attack paths that are crucial to understand threats and then to decide appropriate countermeasures. This is helpful in identifying potential threats, possible attacks, and known vulnerabilities in a cloud system; and (3) the VM protection model: This consists of a VM profiler, a security indexer, and a state monitor. Depending on various factors, such as connectivity, the number of vulnerabilities present, and their impact scores, a security index will be specified for all the VMs in the network.

Regarding the NICE framework, its major components are distributed and lightweight NICE-A software modules in each physical cloud server, a network controller, a VM profiling server, and an attack analyzer. The latter three components are located in a centralized control center connected to the software switches on each cloud server (i.e., virtual switches built on one or multiple Linux software

bridges). NICE-A is a software agent implemented in each cloud server connected to the control center through a dedicated and isolated secure channel. OF tunneling or virtual local area network (VLAN) approaches are used to separate them from the normal data packets. The network controller is responsible for deploying attack countermeasures based on decisions made by the attack analyzer.

The NICE system components include the following:

- NICE-A. This is a network-based intrusion detection system (NIDS) agent installed in each cloud server. It scans the traffic going through Linux bridges that control all the traffic among VMs and in/out from physical cloud servers.
- VM profiling. To get precise information about the state, services running, open ports, etc., of VMs in the cloud, we can build profiles for those items. Connectivity with other VMs is the major factor that counts toward a VM profile.
- Attack analyzer. The major functions of the NICE system are performed by the attack analyzer, which includes procedures such as attack graph construction and update, alert correlation, and countermeasure selection.
- Network controller. This is a key component to support the programmable networking capability to realize the virtual network reconfigurations based on OF protocols.

In Ref. [2], they also gave an idea about NICE security measurement, attack mitigation, and countermeasures. Several countermeasures can be taken to restrict attackers' capabilities. When vulnerabilities are discovered or some VMs are identified as suspicious, it is important to differentiate between compromised and suspicious VMs. The countermeasure serves the purpose of (1) protecting the target VMs from being compromised, and (2) making attack behavior stand prominent so that the attacker's actions can be identified.

The performance of NICE is conducted in two directions in Ref. [2]: the security performance and the system performance. Both have been measured in detail by using virtual environment. The second one demonstrates the feasibility of NICE and shows that the proposed solution can significantly reduce the risk of the cloud system from being exploited and abused by internal and external attackers.

To conclude, NICE used the attack graph model to conduct attack detection and prediction. The proposed solution investigates how to use the programmability of software switches to improve the detection accuracy and defeat the victim exploitation phases of collaborative attacks. NICE only investigates the NIDS approach to counter zombie explorative attacks.

FRESCO: Modular Composable Security Services for SDNs

In OF, we need to dramatically rethink the relationship between the data and control planes of the network device. From a network security perspective, these networks offer researchers with an unprecedented singular point of control over the network flow routing decisions across the data planes of all OF-enabled network components. An OF security application can implement much more complex logic than simply halting or forwarding a flow.

There are a few challenges that motivate us to design a new security scheme in SDNs: (1) information deficiency challenge: OF controllers do not uniformly capture and store TCP session information, among other key state tracking data. However, this is often required to develop security functionality (e.g., TCP connection status, IP reputation). This is referred to as information deficiency challenge. They incorporated a database module, FRESCO-DB, which simplified the storage and management of session state; (2) security service composition challenge: The proposed framework incorporated a modular and composable design architecture, inspired by the Click router architecture. This fosters a rapid and collaborative development of applications through module composition; and (3) threat response translation challenge: The OF protocol enables the controlling software layer to communicate flow handling instructions to the data plane.

The FRESCO framework consists of an application layer (which provides an interpreter and application programming interfaces (APIs) to support composable application development) and a security enforcement kernel (SEK; which enforces the policy actions from developed security applications). Both components are integrated into Nox, an open-source OF controller.

Nox Python modules were used to implement FRESCO's application layer. This is extended through FRESCO's APIs to provide two key developer functions, as follows:

1. FRESCO development environment (DE) and resource controller (RC). They provide FRESCO application developers with OF switch- and controller-agnostic access to network flow events and statistics. Developers use the FRESCO script language to instantiate and define the interactions between Nox Python security modules, which, in turn, invoke FRESCO internal modules. Those modules are instantiated to form a security application driven by the input specified via the FRESCO scripts. These are accessed via FRESCO's DE database API. These instantiated modules are executed by FRESCO DE as the triggering input events are received.
2. FRESCO SEK. Diverse security policies, such as DROP, REDIRECT, and QUARANTINE, can be enforced by security applications developed in FRESCO scripts to react to network threats by simply setting an action variable. These high-level security policies can help developers focus on implementing security applications that are translated into flow rules for OF-enabled switches by FRESCO DE. FRESCO incorporates an SEK, which is integrated directly into the OF controller on which FRESCO operates.

On the implementation perspective of the FRESCO architecture, Python is used to implement the FRESCO application layer prototype and runs as an OF application on Nox. The prototype operates on Nox v.0.5.0 using the OF v.1.1.0 protocol and is implemented in approximately 3000 lines of Python. FRESCO modules are implemented as independent Python objects, and inputs and parameters of a module are input variables to the Python object. The return values of a Python object are considered as output values of a module.

The FRESCO SEK is implemented as a native C++ extension of the Nox source code in approximately 1160 lines of C++ code. The modified OF command function was used to send OF commands to network switches and to capture flow rules from all OF applications.

To evaluate the components in FRESCO, Mininet was implemented, which provided a rapid prototyping environment for the emulation of OF network switches. Using Mininet, they have emulated one OF network switch, three hosts connected to the switch, and one host to operate their Nox controller. Flow generation has been performed by selecting one or two hosts to initiate TCP or UDP connections.

Despite the success of OF, developing and deploying complex OF security services remains a significant challenge. The proposed FRESCO is presented as a new application development framework specifically designed to address this problem. The FRESCO architecture has been integrated with the Nox OF controller and presents several illustrative security applications written in the FRESCO scripting language [3].

Revisiting Traffic Anomaly Detection Using SDN

Small office/home office (SOHO) and purely home networks have had an explosive growth over the last decade because of the widespread penetration of broadband Internet in the home market. Moreover, computers in such networks are often vulnerable. SDN offers a natural opportunity to delegate the task of network security to the home network while sparing the home users a natural opportunity to delegate the task of network security to the home network. Moreover, the home user is spared from complex security management tasks. The authors propose a programmable home network router that provides the ideal platform and location in the network for detecting security problems.

In Ref. [4], four prominent anomaly detection algorithms are implemented in the Nox SDN controller. A detailed accuracy evaluation has been performed on real-world traffic data sets collected at three different network deployment points. Efficiency evaluation of the SDN implementations on home and SOHO network data sets shows that, in addition to providing better accuracy, this approach allows line rate anomaly detection.

Here, the implementations of four prominent traffic anomaly detection algorithms in the context of an SDN are briefly described.

1. Threshold random walk with credit-based (TRW-CB) rate limiting algorithm. It detects scanning worm infections on a host by noting that the probability of a connection attempt of being a success should be much higher for a benign host than a malicious one. The algorithm leverages this observation using sequential hypothesis testing (i.e., likelihood ratio test) to classify whether the internal host has a scanning infection. The algorithm maintains a queue of new connection initiations (i.e., TCP SYNs) that are yet to receive a response (i.e., a SYN/ACK) for each internal host.
2. Rate limiting. This uses the observation that, during virus propagation, an infected machine attempts to connect to many different machines in a short span of time. On the other hand, an uninfected machine makes connections at a lower rate and is more likely to repeat connection attempts to recently accessed machines.
3. Maximum entropy detector. This estimates the benign traffic distribution using maximum entropy estimation. Training traffic is divided into 2348 packet classes, and maximum entropy estimation is then used to develop a baseline benign distribution for each class. Packet classes are derived from two dimensions. The first dimension contains four classes (TCP, UDP, TCP SYN, and TCP reset (RST)). In the second dimension, each of these four classes is split into 587 subclasses based on destination port numbers.
4. Network advertisement (NETAD). This operates on rule-based filtered traffic in a modeled subset of common protocols. The filter removes uninteresting traffic based on the premise that the first few packets of a connection request are sufficient for traffic anomaly detection.

In Ref. [4], the data have been collected at benign traffic at three different locations in the network because the aim was to study the accuracy of anomaly detection algorithms in a typical home network, a SOHO network, and an internet service provider (ISP). To collect the attack traffic, denial of service (DoS) (TCP SYN) and fragile (UDP flood) have been launched simultaneously from three end hosts.

An SDN using OF and Nox allows a flexible, highly accurate line rate detection of anomalies inside home and SOHO networks. The key benefit of this approach is that the standardized programmability of SDN allows the algorithms to exist in the context of a broader framework.

Language-Based Security for SDNs

Analyzing the fundamental problem of how to program SDN in a secure and reliable manner is discussed in Ref. [5]. The solution involves the development of a new programming model that supports the concept of a network slice. The isolation of the traffic of one program from another is achieved with the help of slices. They also isolate one type of traffic from another, within that same program. They have developed a semantics for slices, which illustrates new kinds of formal modular reasoning principles that network programmers can now exploit. It provides definitions of the end-to-end security properties that the slices entail and proves the correctness of a compiler for an idealized core calculus in a slice-based network programming. They have also described their implementation, which is equipped with a translation validation framework that automatically verifies compiled programs using the Z3 theorem prover.

It is challenging today to implement isolation in networks. For this, a large set of devices, including routers, switches, and firewalls, to be manually configured, can be used to block forbidden traffic, but they allow other traffic to traverse the network. Developing and maintaining these configurations will be done using low-level and vendor-specific configuration languages, and this work is tedious for network operators. Simple errors can often lead to serious security breaches. In Ref. [5], it was suggested that using a high-level programming language can make it easy to describe forwarding policies and construct isolated subnetworks while leaving the tedious, error-prone work of generating correct and efficient low-level configurations to a compiler and an SDN. Such an approach seemed to solve the problem that was faced today: networks have traditionally been built out of closed devices that cannot be programmed, except through proprietary and idiosyncratic interfaces. One cannot use Nox as a solution because it lacks mechanisms for isolating the traffic of one module from the traffic of another.

Moreover, we have to see that network programming is insecure and noncompositional. Even the smallest, most trivial modules need to explicitly avoid interfering with other modules whose functionality is completely orthogonal. The programmer determines which traffic will be processed and how it will be forwarded only by analyzing every module in their program. A different approach is described in Ref. [5]. They provided programmers with a high-level, correct-by-construction abstraction that supports the programming-isolated slices of the network, rather than forcing programmers to rely on an external, coarse-grained hypervisor or their own on-off, *ad hoc* techniques for building modular and secure network programs. A slice is defined in terms of the following ingredients: a topology composed of switches, ports, and links, and a collection of predicates on packets, one for each of the outward-facing edge ports in the slice.

Several distinct technical contributions are made [5]. They developed a formal calculus that models a network program as a function from packets to sets of packets to precisely analyze the semantics of slices. The system can execute multiple programs in a single network. The formalized isolation serves as a pair of noninterference conditions: one with respect to a notion of confidentiality and another with respect to integrity. A slice is isolated from another if running them side by side in the same network does not result in a slice leaking packets into the other slice. They defined several intuitive security properties such as isolation and developed an operational condition called separation that implies the isolation property. Finally, they formalized a compilation algorithm and proved that it establishes separation and isolation.

In Ref. [5], a problem that arises in the context of configuring networks is addressed, but how should one express and verify the isolation? This is a fundamental issue. Their solution used technology developed by, and of interest to, the programming languages community: the foundation for their solution is a correct-by-construction programming abstraction. To describe the execution of network programs, they used structured operational semantics. They have used concepts such as confidentiality, integrity, and observational equivalence, drawn from classic work on language-based information-flow security. They used translation validation to obtain assurance.

Moreover, they used familiar proof techniques to validate all of their theorems.

Overall, in Ref. [5], they showed how to define, analyze, implement, verify, and use the slice abstraction to build secure and reliable SDNs. The definition of slices leads to an elegant theory of isolation and proofs of strong end-to-end properties based on observational equivalence, such as confidentiality and integrity. The implementation is highly reliable because they encoded the semantics of their network policies as logical formulae and the use of Z3 theorem prover to validate that their compiler generates outputs that are equivalent to inputs. These encodings also allow them to automatically verify the isolation properties of compiled programs.

In summary, the slice abstraction provides network programmers with a powerful means to seal off portions of their SDN programs from outside interference, such as modules and abstract data types in conventional programming languages. The ability to impose strong boundaries between different program components provides important security benefits and simplifies the construction of programs in settings where security is not an issue. By carving a large program up into slices, a programmer can reason locally about each slice instead of globally when attempting to determine how and where their traffic is forwarded.

Scalable Fault Management for OF

High reliability is an important requirement because a transport connection aggregates traffic. Automatic recovery mechanisms that are triggered by operations, administration, and maintenance (OAM) tools [6] are required for reliability to reestablish connectivity when a path failure occurs.

Recovery is categorized into restoration and protection. For restoration, detour or alternate paths are computed and configured only after a failure is detected. This method is relatively slow. In contrast, for protection, a backup path is configured parallel to the working path. Hence, a fast switch-over minimizes traffic disruption whenever a failure is detected. Transport applications demand a 50-ms recovery time protection to be supported by any transport network

technology. The traffic engineering function can calculate recovery paths in later stages.

In Ref. [6], they proposed to relax the separation of control operations to include connectivity monitoring OAM in the switch to overcome the scalability limitations of centralized connectivity monitoring. The connectivity monitoring must operate in a proactive manner to ensure fast detection of any impairment along the path. The source end of the monitored flow will be the entry point of a transport tunnel. It periodically emits probe packets and interleaves them into the data flow running along that path. The destination end extracts the probe packets from the data flow. If the destination end stops receiving the probe packets for a long enough period (which is referred to as the detection time) or a received packet indicates that the source endpoint detects some errors in the other direction, known as remote defect indication, some node or link in the monitored flow is assumed to have failed, and the destination end node switches over to an alternate path.

They applied the strict integration of the aforementioned functions with the OF switch forwarding to implement an improved fault management efficiently. They proposed to extend the OF v.1.1.0 switch forwarding model with new entities and showed what protocol extensions are essential to provide these novel entities. Nevertheless, they did not intend to provide a comprehensive protocol specification but rather an insight on the necessary switch model and protocol updates.

To generate monitoring messages within the switch, they allow multiple monitored entities to use the same message rate to share a monitoring packet generator. To reduce the amount of processing required in the switch, they separate packet generation from formatting that is filling in identifiers or timers.

Like any other packet, incoming OAM packets enter the switch through ports. First, the monitored tunnel including the OAM packets is identified with a flow table entry. The typical actions associated to that entry are removing the tunnel tags and passing the packet to the next stage, where the payload is further processed. At this phase, demultiplexing is either done as part of popping the tunnel tag or expressed as an additional rule deployed in a later flow table.

The OAM packets can be terminated at a group table entry containing a single action bucket. The content of the OAM packet is

processed, and the timers associated to the OAM termination points are updated. The switch notifies the controller on the occurrence of any monitoring and protection events. To support a generic notification message, an OF error message was modified. Two notification messages are defined here: (1) the messages that report any changes to the status of the monitoring entity and (2) the messages that report the protection reactions. This separation is considered to allow mixed scenarios, where controller-driven restoration is combined with switches that perform continuity monitoring.

1. Fault management in MPLS transport profile. In multiprotocol label switching-transport profile (MPLS-TP) protection, switching is performed by the endpoints of the working path and the backup path. The endpoints may notify each other on protection actions using a data plane–associated protocol, such as Protection State Coordination or a mechanism that is part of the control plane, or using the generalized MPLS (GMPLS) Notify message to stay synchronized. The MPLS-TP OAM supports a continuity check packet flow to detect label switch path (LSP) failures. The recommended implementation is based on the Bidirectional Forwarding Detection (BFD) protocol.
2. Adding BFD and protection support to OF. The current implementation is based on an extension of the OF v.1.0 reference switch implementation that supports MPLS forwarding. As a consequence, they rely on configurable virtual ports instead of using group table entries to implement the monitoring packet construction procedure. This means that not only the packet generator, but also the other packet construction steps are implemented with virtual ports and configured through virtual port management commands.

To evaluate their scheme, they measured the failure time in a testbed consisting of two OF label edge routers (LERs) with two LSPs between them (one working and another as a backup). The switches were implemented as modified OF v.1.0 soft switches on Linux, and the controller was a modified version of the Nox open-source controller running on Linux. Two BFD sessions, both running with a packet transmission interval of 10 ms, monitor the working and backup of LSP by giving a maximum detection time of 30 ms. They transmit

constant bit rate (CBR) traffic with roughly 800 packets per second from the source across the working LSP. The captured inter-arrival time of the CBR packets before and after recovery at the sink is an approximation of the full protection time.

Central control entity is expected to enhance scalability by means of offloading the controller by placing control functions at the switches. By redefining the role of the link-layer discovery protocol (LLDP)-based centralized monitoring, the aforementioned gain can be achieved. As a first step, LLDP is not used any longer for the continuity check, which relaxes the time constraints from milliseconds to seconds. Afterward, the new link detection and failure declaration are decoupled by using switch monitoring features together with link-down notifications. As a consequence, there is no need to consider accidental packet losses and jitters, which further increases packet sending intervals.

In conclusion, they proposed to slightly relax the separation of control and forwarding operations to overcome the scalability limitations of centralized fault management. To provide a scalable way for data plane connectivity monitoring and protection switching, they argue that OAM is a function that needs to be distributed and deployed close to the data plane. They propose to place a general message generator and processing function on OF switches. They describe how the OF v.1.1.0 protocol should be extended to support the monitoring function. Moreover, they prove that data plane fault recovery within 50 ms can be reached in a scalable way with their proposed extensions, through their experiments.

A Dynamic Algorithm for Loop Detection in SDNs

The existence of loops, which are cyclical paths through the network's switches, which can cause some packets to never leave the network, will be a potential problem in computer networks. In Ref. [5], a dynamic algorithm that is built on header space analysis is presented, which allows the detection of loops in SDNs like the ones created using OF over a sequence of rule insertions and deletions on the network's switches, and the key ingredient in the algorithm is a dynamic, strongly connected component algorithm. In the article, the network model has been illustrated as a directed graph because it will be easier

Table 16.2 Comparison of the Schemes Discussed

SCHEME SOURCES	[2]	[3]	[4]	[5]	[6]	[7]	[8]
Uses OF/Nox (Y/N)	Y	Y	Y	Y	N	Y	Y
Introduce new architecture based on OF/Nox	Y	N	Y	N	N	Y	Y
Experiments (E) or real cases (R)	E	E	E	E	E	E	E
Software (S) or hardware (H)	S	S	S	S	S	S	S
Introduces new language for SDN (Y/N)	N	N	N	N	Y	N	N

to understand. Hence, concepts of header space analysis have been translated into the language of graph theory.

1. Rule graphs and the dynamic loop detection problem. In Ref. [7], they show how to model a network in the same way as a directed graph. Through this, the translation can introduce notions from dynamic graph algorithms to help one compute port-to-port reachability in their network over rule updates in an efficient manner.
2. A dynamic, strongly connected component algorithm. It introduces an algorithm that allows to dynamically track the strongly connected components (SCCs) in a graph over a sequence of edge insertions and deletions. It also shows how to use this dynamic algorithm to solve the dynamic loop detection.

Discussion

A comparison of all the aforementioned SDN security schemes is presented in Table 16.2.

Conclusion

In conclusion, SDN is an emerging technology that allows for granular security by giving complete control of the network to the administrator. The controller is the brain of SDN, and without proper security wrapped around the controller, the network becomes completely vulnerable to accidental changes or malicious attacks. There are different approaches that can be used to achieve this task and take the full benefit of this new technology.

17

Investigation of Anycast Implementation in Software-Defined Networking

Contents

Introduction 436
Related Work 438
Preliminaries of SDN and OpenFlow 440
Implementation of Anycast Service Based on OpenFlow in Intradomain Environments 441
  The System Architecture 441
  The Design of the Anycast Controller 442
    Information Gathering Module 443
    Routing Decision Module 444
    Address Resolution Module 446
    Data Transmission Module 447
Numerical Results and Analysis 447
Extension of Anycast Implementation in Interdomain Environment 452
The Merit of the Proposed Anycast Implementation Strategy in the Interdomain Environment 453
Conclusion 454
Acknowledgments 455
References 455

Introduction

Anycast is a paradigm of communication for service discovery, which selects the best one of the service providers in an anycast group as a destination in a robust and efficient manner, which has been an important service model adopted in various networks for diverse applications. Anycast technologies are widely used in content delivery networks (CDNs) for the large-scale distribution of content on the Internet and the direction of requests to find the desired content [1]. To avoid the substantial delays because a transmitting node needs to wait for its next-hop relay node to wake up, wireless sensor networks (WSNs) adopted an anycast-based scheme for each node to opportunistically forward packets to the first neighboring node that wakes up among multiple candidate nodes [2]. Mobile *ad hoc* networks (MANETs) also used anycast-like proposal to shorten the transmission paths between the requester and service providers and reduce the amount of request and reply packets [3].

Currently, application-layer anycast and network-layer (or IP) anycast are the two main research directions on anycast communications and have been widely used in many scenarios because of its inherent ability of service discovery. However, the nonawareness of the topology changes and load conditions hampers the deployment of application-layer anycast [4]. In addition, the address translation of application-layer anycast is costly when the network is under heavy load [5]. IP anycast overcomes these issues with simple implementation but needs to modify existing routing protocols with new router configurations to support anycast service, and thus, the anycast server selection process and the packet routing process are accomplished in the switching equipment, resulting in higher efficiency and robustness than that of application-layer anycast [6]. However, the existing network equipment (e.g., routers and switches) act as black boxes, leading to the poor scalability for the deployment of IP anycast. Furthermore, because of its inherent nature, IP anycast makes the address nonaggregatable in the routing table [7].

Software-defined networking (SDN) has been proposed to programmatically control networks by decoupling the control from the data plane, which is a promising technique to lower the barrier for deploying and managing new functionality, applications, and services

in the networks. Thus, the preceding issues encountered in the wide deployment of application-layer anycast and IP anycast can be readily solved by this new paradigm of architecture. The main thrusts in SDN are OpenFlow [8] by the Open Networking Foundation (ONF); Protocol Oblivious Forwarding (POF) [9] by Huawei Technologies Co., Ltd.; PEARL [10] by the Institute of Computing Technology, Chinese Academy of Sciences; etc. OpenFlow is currently the most promising and popular realization of SDN and has been commercially produced by CISCO, HP, Juniper, NEC, etc. In OpenFlow, the software running at a logically centralized controller, manages a collection of switches hosting programmable forwarding tables. In an effort to ease the development and deployment of anycast service in the Internet, this chapter makes the following main contributions:

- This chapter presents a new load-aware anycasting based on OpenFlow technology for intradomain environments and develops the Information Gathering Module, Routing Decision Module, Address Resolution Module, and Data Transmission Module to support anycast service and load-aware mechanism at the controller.
- Extensive Mininet experiments are conducted to validate the effectiveness and accuracy of the proposed OpenFlow-based anycast scheme. The results demonstrate that the performance of the developed load-aware anycast scheme outperforms that of existing solutions in terms of anycast request delay and loss probability.
- The developed OpenFlow-based anycast scheme for the intradomain environment is then extended to the solution for interdomain networks of global deployment strategies. The analysis shows that this strategy can be adopted as an evolvable way for the large-scale deployment of load-aware anycast service in the current Internet.

The remainder of this chapter is organized as follows. "Related Work" presents the existing studies on anycast implementation and the issues encountered. The preliminaries of SDN and OpenFlow are shown in "Preliminaries of SDN and OpenFlow." "Implementation of Anycast Service Based on OpenFlow in Intradomain Environments"

describes the design of load-aware anycasting based on OpenFlow for a single autonomous system (AS), including the system architecture and, particularly, the design of the OpenFlow controller, with Information Gathering Module, Routing Decision Module, Address Resolution Module, and Data Transmission Module to support anycast service. The extensive Mininet experiments and analysis are conducted in "Numerical Results and Analysis." "Extension of Anycast Implementation in Interdomain Environment" then extends the proposed scheme to the solution for the interdomain of multidomain scenarios to support evolvable deployment of load-aware anycast service in a global environment. This chapter ends with the "Conclusion."

Related Work

The existing studies on the investigation of anycast service can be classified into two categories: application-layer anycast and IP anycast. For example, Ma et al. [11] targeted the challenges of anycast implementation in the large-scale global environments and managed to solve the problems of scalability to worldwide implementation, anycast query latency minimization, and optimal server selection strategies. The authors in Ref. [12] presented a context-aware anycast multimedia provisioning scheme in the application layer by using the distributed deployed service registry nodes that collect and maintain the server's contexts and content descriptions, and perform the mapping of the anycast address of the client request to the unicast address of the most convenient server based on the contexts of the clients and servers. Bhattacharjee et al. [13] designed a special name structure for application-layer anycast service and adopted a resolver to translate anycast address to IP address. The communication procedure is similar with the domain name system (DNS) resolution. The proposed scheme is simple and easy to be deployed, but it is nonsensitive to the topology and load status changes. The authors in Refs. [14] and [15] used the statistic and stochastic methods in the scheduling of the anycast manager, which improves the performance of application-layer anycast. Garcia-Luna-Aceves and Smith [16] proposed a mechanism to accelerate the process of anycast address translation, which alleviates the performance degradation resulted by address translation, but this scheme cannot solve the

problem radically. The authors in Ref. [17] investigated the problem of directing clients to the replicated servers and presented an anycasting paradigm at the application layer by providing a service that uses an anycast resolver to map an anycast domain name and a selection criteria into an IP address.

On the other hand, the main focus of IP anycast is the design of routing strategies and the improvement of protocols to support anycast service in the current IP architecture. For example, Alzoubi et al. [18] presented a practical architecture for load-aware IP anycast based on the traditional route control technology and its usage in CDN environments. The proposed scheme made use of route control mechanisms to consider server and network load to realize load-aware anycast. The authors in Ref. [19] developed a distributed system that can offload the burden of replica selection while providing these services with a sufficiently expressive interface for specifying mapping policies. The prototyping of the system supports the IP anycast and can handle many customer services with diverse policy objectives. Katabi and Wroclawski [20] proposed an infrastructure to achieve global deployment of IP anycast, which was extended by Ballani and Francis in Ref. [6]. Lenders et al. [21] proposed a density-based anycast routing strategy to improve the stability of IP anycast. The authors in Ref. [22] proposed a new anycast communication model based on IPv6 to solve the problems of scalability and communication errors between clients and servers. A proxy-based architecture that provides the support for stateful anycast communications, while retaining the transparency offered by the native anycast, was proposed in Ref. [23] to overcome the routing scalability issues and the lack of stateful communication support in the traditional IP anycast.

To the best of our knowledge, there is little research conducted in the current literature to investigate the performance of anycast service using OpenFlow. Very recently, Othman and Okamura [24] proposed a content anycast solution using OpenFlow to improve the effectiveness of content distribution. However, this study only presented a mapping mechanism between each file and its associated identifier (ID) and redirected the content requests that cannot be handled by one server to another, rather than solving the inherent problems resulting from the traditional anycast service.

SDN [25] is a promising technique to lower the barrier for deploying and managing new functionality, applications, and services in the networks. OpenFlow [26] was raised in 2008 by the Clean State Team of Stanford University, and it has been the promising and popular technology to realize the SDN because of its ability to support the decoupling of the control plane and the data (forwarding) plane. OpenFlow centralizes the control of the flow table in the switch devices to an external programmable and flexible controller, and provides a secure protocol to facilitate the communication between the controller and switch devices. OpenFlow has been commercially deployed in data center networks and wide area networks such as Google.

A packet is forwarded by the switches based on the entries in the flow table. OpenFlow switches possess a much simpler flow table than ordinary switches. The flow table in the OpenFlow switch consists of many flow entries, each of which includes six parts: Match Fields, Priority, Counters, Instructions, Timeouts, and Cookie, as shown in Figure 17.1. The Match fields is used to match against packets, which consists of ingress port and packet headers; the Priority is adopted for matching the precedence of the flow entry; the Counters is used to update for matching packets; the Instructions is used to modify the action set or pipeline processing; the Timeouts sets the maximum amount of time or idle time before flow is expired by the switch; and the Cookie is the opaque data value chosen by the controller, which can be used by the controller to filter flow statistics, flow modification, and flow deletion.

The standard v1.3 of OpenFlow supports 40 match fields, where 10 match fields are required to be supported by a switch, including Ingress Port, Ethernet Source Address and Destination Address, Ethernet Type, IPv4 or IPv6 Protocol Number, IPv4 Source Address and Destination Address, IPv6 Source Address and Destination Address, transmission control protocol (TCP) Source Port and

Match fields	Priority	Counters	Instructions	Timeouts	Cookie

Figure 17.1 Main components of a flow entry in a flow table. (From OpenFlow switch specification. Available at http://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf.)

Destination Port, and user datagram protocol (UDP) Source Port and Destination Port [26].

Implementation of Anycast Service Based on OpenFlow in Intradomain Environments

The System Architecture

The proposed architecture for an anycast system based on OpenFlow technology consists of anycast servers providing anycast service, the anycast client, OpenFlow-enabled anycast (OFA) switches, and the anycast controller, where anycast servers and anycast clients are directly or indirectly connected by OFA switches. In this section, we consider the case for the intradomain networks of a single AS, where the OFA switches are deployed in the subnetworks of an AS and are interconnected through IP tunnel or dedicated link to form a wide-area layer 2 network (see Figure 17.2). The anycast controller has IP connections with OFA switches.

The anycast controller is the key component in this system, with the aim of assigning and recycling anycast addresses and making the reasonable and appropriate routing decisions. To increase the scalability of supporting more anycast service, we apply a set of addresses for anycast services. The anycast controller can collect the status and information of OFA switches and the connected anycast servers/

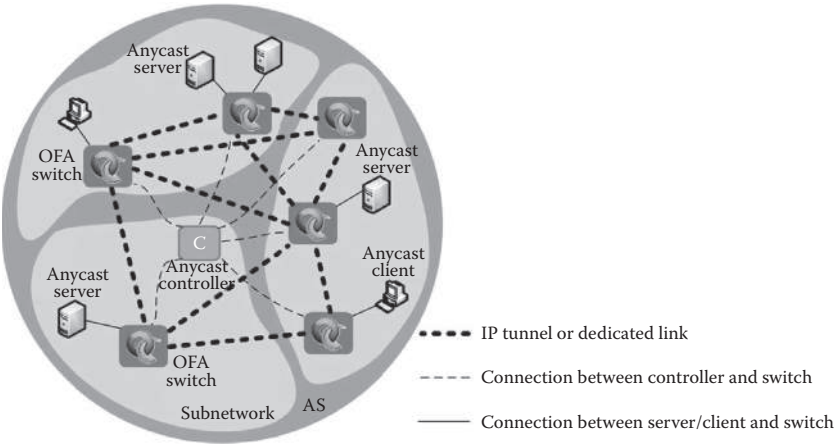


Figure 17.2 System architecture of OpenFlow-based anycasting in an AS.

clients, and make the routing decisions for anycast requests according to various performance metrics. The routing decisions are added to the OFA switches as the entries in the flow table. The OFA switches forwards the packets according to the entries in the flow table. In addition, the OFA switches possess the ability to advertise the prefix of anycast addresses.

The Design of the Anycast Controller

OpenFlow achieves the higher flexibility in the routing of network flows and the freedom to change the behavior of network traffic by separating the control plan in network devices from the data plane, resulting in the importance of the controller in OpenFlow technologies. In this section, the controller will be designed to support anycast service, as shown in Figure 17.3. Many kinds of OpenFlow controller system have been widely reported, such as POX, NOX, Maestro, Beacon, simple network access control (SNAC), Floodlight, etc. Because of the

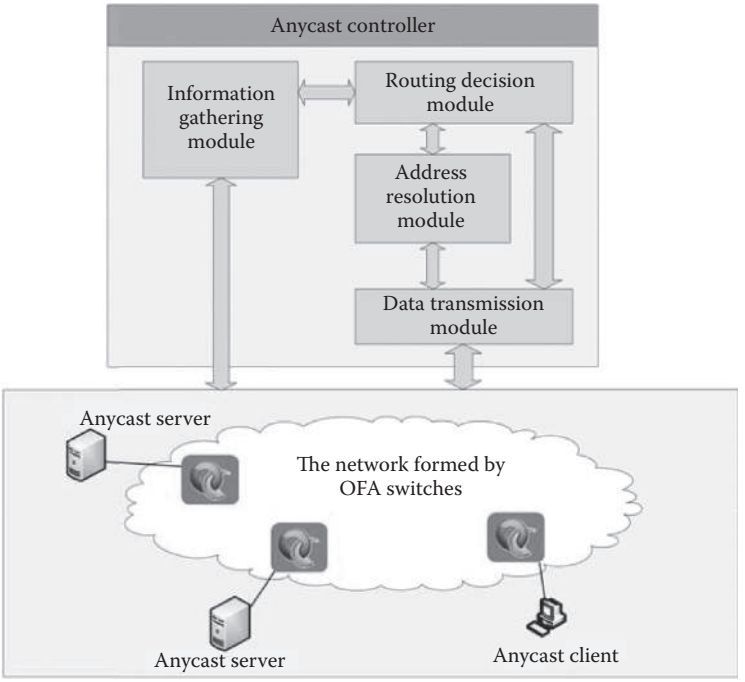


Figure 17.3 Design of the anycast controller.

feature of rapid development and prototyping of network control software, in this chapter, we develop the anycast controller based on the POX platform [27] and particularly design the Information Gathering Module, Address Resolution Module, Routing Decision Module, and Data Transmission Module for anycast service. In what follows, the detailed design of these modules will be presented.

Information Gathering Module The anycast controller is responsible for being aware of the topology changes and the status changes of loads at the anycast servers/clients and network links to make appropriate decisions of routing strategies. If the server selection in anycast service depends on the metrics of hop-count from the anycast client who originates the requests to the anycast server, the anycast controller must be aware of the network topology. On the other hand, if the load balance needs to be considered in routing decision making, the load status of each network link and anycast servers and clients should be known by the controller.

The anycast controller possesses the abilities to detect the network topology and load status of anycast servers and clients and network links by communicating with OFA switches based on the OpenFlow protocol [26].

- The OFA switches report the changes of ports to the anycast controller when receiving a Port-Status message of asynchronous type. This can be used by the anycast controller to detect the topology changes of the underlying networks.
- The anycast controller can get the status information of OFA switches by sending a Read-State message of controller-to-switch type. Keeping track of the status statistics of all OFA switches, the anycast controller can be aware of the required information, for example, load status on network links, of the whole network.

For the implementation, we can set a timer in the anycast controller to trigger a request for the required information to OFA switches periodically. According to the response of the OFA switches, the anycast controller can be aware of the topology of the whole network and monitor the load status of each link and port, which is helpful to

support routing decision making, stated in the next subsection, under different measurement metrics.

Routing Decision Module This module makes routing decisions for packets that fail to match the entries in the flow tables of OFA switches. In the proposed controller, two routing metrics are considered: hop counts and link loads. In what follows, the implementation of the two strategies in the routing decision module will be presented.

The Routing Strategy Based on Hop Counts The routing strategy based on the hop counts choose the nearest anycast server to serve the request, where the term *nearest* is calculated by the hop counts from the anycast server and the anycast client who originates the request. To achieve this purpose, it is necessary to calculate the hop counts between any two OFA switches in the network. If the entire network topology is abstracted as a graph, then the problem is transformed into calculating the shortest path between any two points in the graph.

The Floyd-Warshall algorithm has been widely adopted in the literature [28–30] for the fundamental graph problem of all-pairs shortest path, which is then used in this chapter to carry out the shortest path calculation between two OFA switches. In the Floyd-Warshall algorithm, there are two possible shortest paths from Point A to Point B in the graph. One is the path that connects Point A and Point B, and the other is the path from Point A to Point B through some other points, say, Point C. Let $Distance(A, B)$ represent the shortest path from Point A to Point B, and let P denote the set of all points in the graph. Thus, for $\forall C \in P$, $C \neq A$, $C \neq B$,

$$Distance(A, C) + Distance(C, B) > Distance(A, B).$$

Then, once traversing all points in the graph, $Distance(A, B)$ records the shortest path from Point A to Point B.

The Routing Strategy Based on Link Loads The routing strategy based on link loads choose the best anycast server to serve the request, where the term *best* means that the link between the anycast client and the anycast server has the optimal loads. The link loads can be measured as the number of packets transferred through the link per time unit,

which can be obtained by the counters of flow table entries in the OFA switches [26] as specified in "Preliminaries of SDN and OpenFlow," and thus can be captured by the anycast controller through the OpenFlow protocol between the controller and the switches.

In the design of the anycast controller, we also adopt the Floyd-Warshall algorithm [30] to perform routing strategies based on the link loads, where the difference from the routing scheme based on the hop counts is that the links in the network topology are weighted with the loads. Algorithm 1 shows the calculation of the path using the routing strategy based on link loads.

Algorithm 1: The calculation of the path based on link loads

Input: *Load_matrix*[*N*][*N*] // representing the link load between node *i* and node *j*
Output: *Path*[*N*][*N*] and *Output_port*[*N*][*N*] // *Path*[*i*][*j*] denotes the path between node *i* and node *j*; *Output_port*[*i*][*j*] represents the output port at node *i* if the packet destination is node *j*

```
1.  N is the number of OFA switches;
2.  Initialization Output_port[N][N];
3.  For i in the range (1, N)
4.      For j in the range (1, N)
5.          If there exists a link between node i and node j
6.              Output_port[i][j] = get_port(link(i, j));
7.          Else
8.              Output_port[i][j] = NULL;
9.          EndIf
10.     EndFor
11. EndFor
12. Path = Load_matrix;
13. For k in the range (1, N)
14.     For i in the range (1, N)
15.         For j in the range (1, N)
16.             If Path[i][j] > Path[i][k] + Path[k][j]
17.                 Path[i][j] = Path[i][k] + Path[k][j];
18.                 Output_port[i][j] = Output_port[i][k];
```

```
            EndIf
        EndFor
    EndFor
EndFor
Return Path[N][N] and Output_port[N][N]
```

Address Resolution Module The main task of the Address Resolution Module is to generate Address Resolution Protocol (ARP) response packets for anycast requests. Specifically, the anycast controller extracts anycast IP address from the ARP request packets and invokes the Routing Decision Module to select a destination anycast server. The Routing Decision Module then returns the Medium Access Control (MAC) address of the selected anycast server and generates an ARP response packet using the destination MAC address. Finally, the Address Resolution Module invokes the Data Transmission Module to deliver the ARP response to the anycast client.

An anycast client should convert an IP address into a MAC address before communicating with anycast servers in the same AS. In particular, according to the ARP protocol, an IP address can be resolved into only one MAC address in an AS. With an anycast service, there may be more than one anycast servers that share the same anycast address in an AS, and thus, an anycast client who originates an ARP request may receive multiple ARP responses, resulting in ARP resolution collision.

The ARP resolution collision can be resolved by changing the way that ARP requests are sent. In anycast service, only one server is selected to respond to an anycast request, and thus, it does not require all the anycast servers to respond to the ARP request. With the help of the anycast controller, the OFA switches along the path between the anycast client and the given anycast server can be built. The anycast controller can then respond to the ARP request on behalf of the selected anycast server. To avoid the collision, this method ensures that only one ARP response would be generated in the process of ARP resolution.

The entry in the flow table of all OFA switches for the ARP request packet should be added previously to direct this kind of packet to the anycast controller, which will choose an appropriate anycast server with the help of the Routing Decision Module, and then generate

an ARP response packet using the selected anycast server's MAC address as the MAC source address of the ARP response packet to send back to the anycast client. Under this circumstance, there is only one ARP response packet sent to the anycast client from multiple anycast servers to accomplish an ARP resolution.

Data Transmission Module The Data Transmission Module is a middleware between OFA switches and the Routing Decision Module and the Address Resolution Module. It receives packets from the OFA switches and transfers the packets into different modules, and vice versa.

The data packets that fail to match the entries in the flow table of an OFA switch will be delivered to the anycast controller. The Data Transmission Module resolves the header of this packet and, according to its protocol type, delivers the packets to the corresponding modules for the further process, as shown in Figure 17.4.

Numerical Results and Analysis

To evaluate the effectiveness and accuracy of the proposed scheme, we designed the anycast controller using POX and implemented the OFA switches and anycast servers/clients in Mininet [31]. POX is a piece of SDN ecosystem [27], which is a platform used for the rapid development and prototyping of network control software using Python. Mininet is a network emulator that runs a collection of clients/servers, switches, and network links on a Linux kernel. It adopts lightweight virtualization to make a single system over the physical network. The clients/servers in Mininet behaves just like a real machine: one can secure shell (SSH) to it and run arbitrary programs.

In this chapter, we consider the simulation environment with the topology of 100 OFA switches in a 10 × 10 grid structure, and each switch connects with a terminal, where we choose 5% of the terminals as anycast servers, and the others act as anycast clients who originate requests. A POX running on another machine acts as the anycast controller. OFA switches, anycast clients/servers, and anycast controllers are connected by virtual links (VLs) that are generated by Mininet.

Anycast request delay and request loss probability are the two key performance metrics adopted to evaluate the effectiveness and

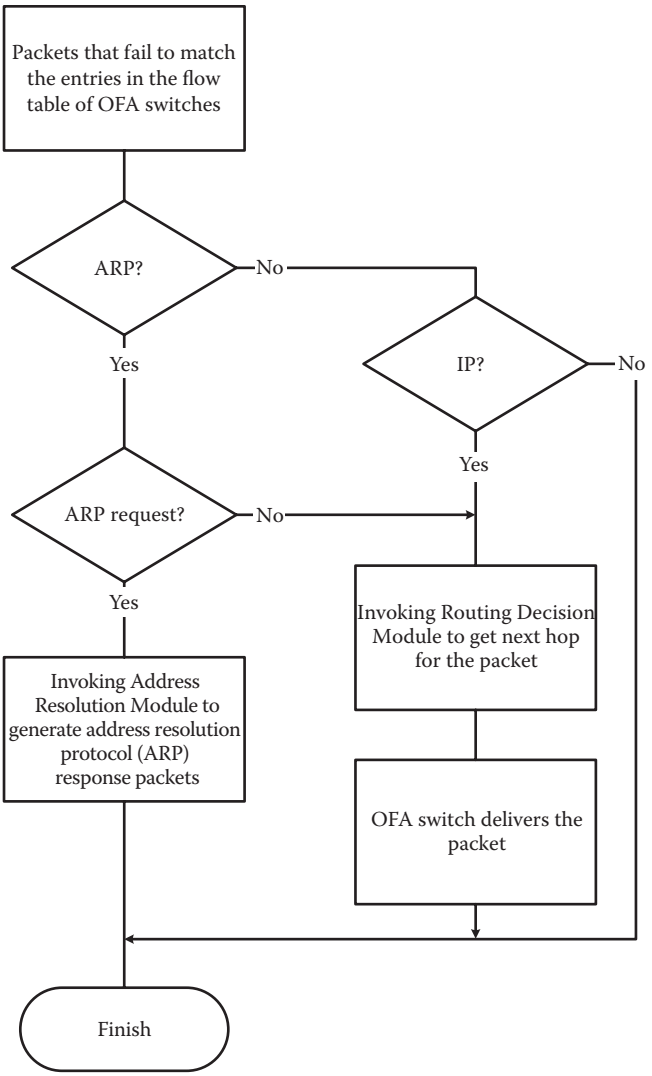


Figure 17.4 Data transmission procedure.

accuracy of the proposed load-aware anycast scheme. Extensive simulation experiments are conducted under various combinations of the number of OFA switches, the number of anycast clients/servers, the bandwidth of VLs, and the status of background traffic. Each simulation was run until the network reaches the steady state. However, for the sake of specific illustration and without loss of generality, the results are presented based on the parameters shown in Table 17.1.

Table 17.1 System Parameters of the Simulation

SYSTEM PARAMETERS	VALUE
Physical machine (PM)	2.5-GHz Intel Core i5 processor 4-GB 1600-MHz memory
Operating system (OS) of PM	Mac OS X 10.7.5 Lion
Virtual machine (VM)	Oracle VirtualBox 4.2.4 r81684
VM image	Official Mininet 2.0.0 [32]
No. of OFA switches	100
No. of anycast servers	5
No. of anycast clients	95
VL type	Mininet TCLink
VL bandwidth	1 Mbps, 2 Mbps
Background traffic	UDP
Queue size of OFA switches	64 packets
Queue size of anycast clients	64 packets
Packet size	64 B

Figure 17.5 depicts the anycast request delay and request loss probability predicated by the existing hop-based scheme and proposed load-aware scheme against the request generation rate under 1-Mbps bandwidth of the VL. As can be seen from the figure, increasing the request generation rate results in the higher request delay and request loss probability. In addition, as the request generation rate increases, the proposed load-aware anycasting has the lower request delay and loss probability in comparison with those predicted by the existing

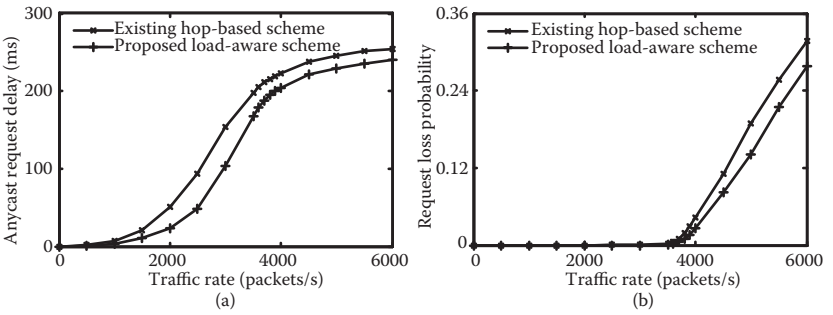


Figure 17.5 (a) Anycast request delay and (b) request loss probability predicated by the existing hop-based scheme and proposed load-aware scheme against a traffic rate under 1-Mbps bandwidth without background traffic.

hop-based anycast, especially under a moderate and higher request generation rate.

Figure 17.6 depicts the anycast request delay and request loss probability for both the hop-based anycast scheme and the proposed load-aware anycast scheme against the request generation rate under 2-Mbps bandwidth of VLs and 0.5-Mbps background UDP traffic to mimic the behavior of networks running for a certain period. The results reveal the similar phenomenon as shown in Figure 17.5, although the network has run for some time. The results also emphasize that, as the loads of the network are going moderate, the load-aware anycasting has significant advantage on the request delay and loss probability.

To further evaluate the merit of the proposed load-aware anycast scheme, we consider the case that some VLs are congested in the topology of 100 OFA switches in a 10 × 10 grid structure with 2-Mbps bandwidth of VLs, and each switch connects with a terminal where we choose 2 terminals (i.e., the upper left corner and the bottom right corner shown in Figure 17.7) as anycast servers, and the other 98 terminals act as anycast clients. In particular, we consider the three cases shown in Table 17.2 with VLs 1, 2, and/or 3 added with 2-Mbps background UDP traffic to mimic the congested status of the link. To this end, Figure 17.8 presents the anycast request delay and request loss probability predicted by the existing hop-based anycast scheme and load-aware anycast scheme under the three cases shown in Table 17.2, and the anycast request generation rate is set

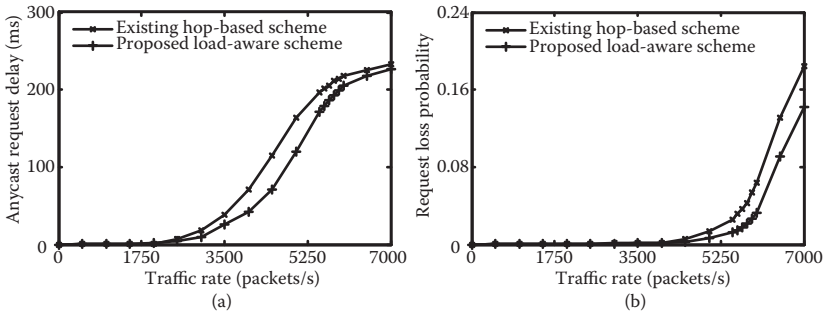


Figure 17.6 (a) Anycast request delay and (b) request loss probability predicated by the existing hop-based scheme and proposed load-aware scheme against a request generation rate under 2-Mbps bandwidth and 0.5-Mbps background UDP traffic.

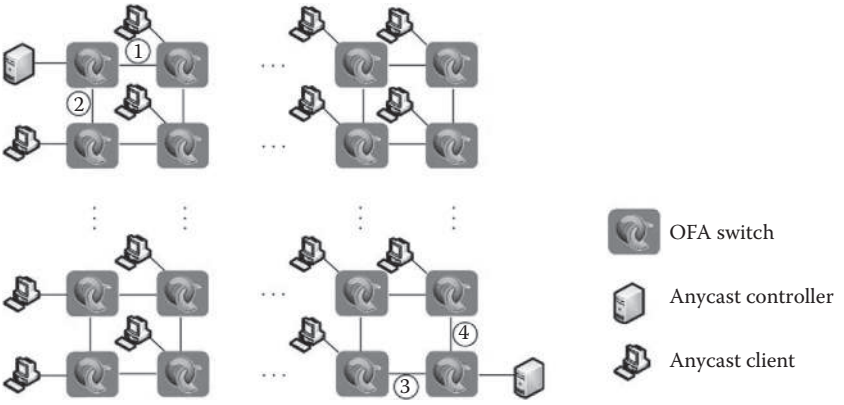


Figure 17.7 Network topology in the 10 × 10 grid structure of OFA switches for performance analysis.

Table 17.2 Three Cases Considered for Performance Analysis

CASES	DESCRIPTION
Case I	Add 2-Mbps background UDP traffic on VL 1
Case II	Add 2-Mbps background UDP traffic on VLs 1 and 2
Case III	Add 2-Mbps background UDP traffic on VLs 1, 2, and 3

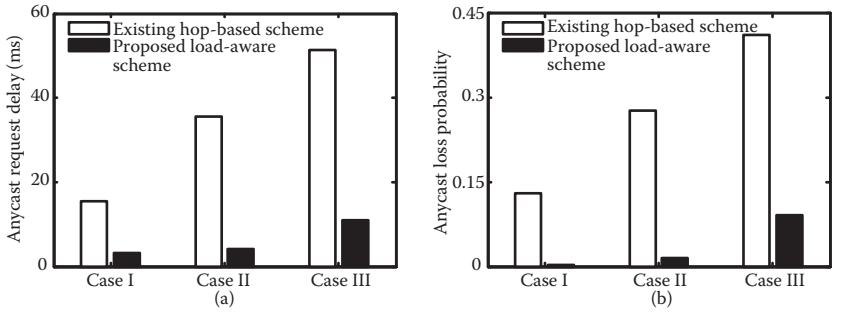


Figure 17.8 (a) Anycast request delay and (b) request loss probability predicated by the existing hop-based scheme and proposed load-aware scheme against the three cases shown in Table 17.2.

to be 200 packets per second. From the figure, we can find that the anycast request delay and loss probability predicted by the existing hop-based scheme are greater than those of the proposed load-aware scheme because the hop-based anycasting encountered the congested VLs 1, 2, and/or 3, whereas the load-aware anycast scheme can bypass the congested link and leverage the light-load link to forward packets.

In addition, the performance of the load-aware anycast scheme under Case II does not degrade too much in comparison with that under Case I. This is because at least two alternative links can be shared by the load-aware scheme under these two cases, which alleviates the burden of each VL. In contrast, the performance of the load-aware anycasting under Case III has significant degradation compared with that of Case I and Case II.

Extension of Anycast Implementation in Interdomain Environment

The anycast has been viewed as a powerful packet addressing and delivery model, and its implementation in the interdomain environment has been widely reported [6,20,33]. In this section, the proposed intradomain anycasting scheme will be extended to the case of interdomain environments with the gradual deployment mode.

In particular, we consider three cases of the configuration and deployment of OFA switches and anycast servers at each AS shown in Figure 17.9: (1) the AS with the deployment of OFA switches and the connected anycast server (see AS 1 and AS 2 in Figure 17.9); (2) the AS with the deployment of OFA switches only (see AS 3 in Figure 17.9); and (3) the AS without the deployment of any OFA switch and anycast server (see AS 4 in Figure 17.9).

The anycast requests originated by an anycast client need to be directed to an OFA switch. In this chapter, we consider the cases that (1) the anycast clients are located at the AS with one or more OFA switches (see AS 3 in Figure 17.9) and (2) the anycast clients are located at the AS without the OFA switch (see AS 4 in Figure 17.9). For both cases, the OFA switches advertise the address prefix for anycast service. Note that only one OFA switch makes the advertisement if multiple switches exist in an AS. For the former case, according to the address prefix advertisement by the OFA switches, the packets with the anycast address as the destination can be directed to the OFA switch in the AS. For the latter case, the anycast requests will be directed to the Border Gateway Protocol (BGP) router in the AS, and the BGP router can then, according to the address prefix advertisement, find the AS that possesses the OFA switches. The anycast requests will then be directed to the BGP router of that AS, and also,

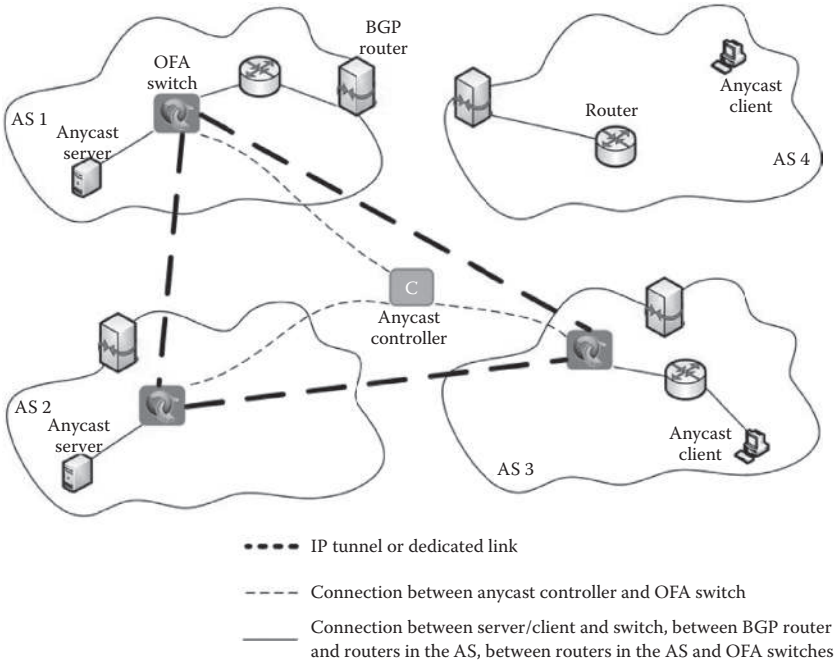


Figure 17.9 Implementation of anycast service based on OpenFlow in the interdomain environment.

according to the address prefix advertisement in that AS, the anycast request can be finally directed to the OFA switch.

Once they reach the OFA switch, the anycast requests will be directed to the anycast server according to the entries in the flow table of the OFA switch. If the anycast requests cannot be matched against the entries in the flow table, they will be directed to the anycast controller who can calculate the required routing path based on the design mechanism stated in "The Design of the Anycast Controller" and add the corresponding entries in the flow table of the OFA switches along the path to the anycast server.

The Merit of the Proposed Anycast Implementation Strategy in the Interdomain Environment

In this section, the proposed scheme will be analyzed in terms of flexibility, feasibility, and scalability. The DNS is a typical example

of anycast implementation in interdomain environments. Because the DNS has been widely deployed, its flexibility and scalability has been validated. Therefore, we mainly compare the flexibility between these two schemes.

- Flexibility. Different anycast services should provide diverse strategies on load-aware routing for different scenarios. However, such consideration is quite difficult to be realized for DNS. In contrast, with the proposed interdomain anycast implementation, the strategies for the different load-aware requirements can be readily realized at the anycast controller with a low cost.
- Feasibility. The proposed strategy does not need to change the current network protocols and architecture; however, it needs to deploy OFA switches in the current Internet to gradually support anycast service.
- Scalability. With a new anycast service, the proposed scheme only needs to provide an address in the anycast address space, establish the connection between new anycast servers and the OFA switches, and add the corresponding new entries in the flow tables of OFA switches for this new anycast service by the anycast controller.

Conclusion

This chapter has investigated the anycast implementation in SDN/OpenFlow environments and presented a load-aware anycasting implementation in the OpenFlow networks of intradomain environments. Extensive Mininet experiments have been conducted to validate the effectiveness and accuracy of the proposed load-aware anycast scheme. The results have demonstrated that the performance of the developed anycasting outperforms that of existing solutions in terms of anycast request delay and loss probability. The developed OpenFlow-based anycast scheme for intradomain environments has then been extended to a solution for the interdomain global deployment strategies. The analysis has shown that this strategy can be adopted as an evolvable way for the large-scale deployment of load-aware anycast service for the current Internet.