Chapter No. 1
"Introducing OpenFlow"
In this package, you will find:
A Biography of the author of the book
A preview chapter from the book, Chapter NO.1 "Introducing OpenFlow"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Siamak Azodolmolky received his Computer Engineering degree from Tehran University and his first MSc. degree in Computer Architecture from Azad University in 1994 and 1998 respectively. He was employed by Data Processing Iran Co. (IBM in Iran) as a Software Developer, Systems Engineer, and as a Senior R&D Engineer during 1992-2001. He received his second MSc. degree with distinction from Carnegie Mellon University in 2006. He joined Athens Information Technology (AIT) as a Research Scientist and Software Developer in 2007, while pursuing his PhD degree. In August 2010, he joined the High Performance Networks research group of the School of Computer Science and Electronic Engineering (CSEE) of the University of Essex as a Senior Research Officer. He received his PhD (with ‘cum laude’) from the Universitat Politècnica de Catalunya UPC in 2011. He has been the technical investigator of various national and EU funded projects. Software Defined Networking (SDN) has been one of his research interests since 2010, in which he has been investigating the extension of OpenFlow towards its application in core transport (optical) networks. He has published more than 50 scientific papers in international conferences, journals, and books. Currently, he is with Gesellschaft für Wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG) as a Senior Researcher and has lead SDN related activities since September 2012. He is a professional member of ACM and a senior member of IEEE.

For More Information:
Whenever I reach the end of a book production, once again I realize that nobody is perfect. I would like to thank the technical reviewers for providing me with fruitful and constructive feedback. Any remaining errors are, of course, my own. I would also like to thank the Packt Publishing team who has been really supportive in getting this book off the ground. The knowledge, support, and experience of many colleagues in the SDN community have been instrumental in filling the gaps in my understanding of SDN. This book was not simply possible without them.

Finally, sincere and especially heartfelt thanks go out to my son, Parsa Azodolmolky. His patience during writing time, while being away from me is greatly appreciated. I love you Parsa.
Software Defined Networking with OpenFlow

Decoupling the network control out of the networking devices is the common denominator of Software Defined Networking (SDN). SDN is a recent paradigm shift in computer networking, where network control functionality (also known as control plane) is decoupled from data forwarding functionality (also known as data plane) and furthermore the split control is programmable. The migration of control logic, which used to be tightly integrated in networking devices (for example, Ethernet switches) into accessible and logically centralized controllers, enables the underlying networking infrastructure to be abstracted from an applications point of view. This separation paves the way for a more flexible, programmable, vendor-agnostic, cost effective, and innovative network architecture. Besides the network abstraction, SDN architecture will provide a set of Application Programing Interfaces (APIs) that simplifies the implementation of common network services (for example, routing, multicast, security, access control, bandwidth management, traffic engineering, QoS, energy efficiency, and various forms of policy management). As a result, enterprises, network operators, and carriers gain unprecedented programmability, automation, and network control, enabling them to build highly scalable, flexible networks that readily adapt to changing business needs. OpenFlow is the first standard interface designed specifically for SDN, providing high performance, granular traffic control across multiple networking devices. This book looks at the fundamentals of OpenFlow, as one of the early implementations of the SDN concept. Starting from OpenFlow switches and controllers up to the development of OpenFlow-based network applications (Net Apps), network virtualization, OpenFlow in Cloud Computing, and a summary of active OpenFlow related open source projects are topics, which are covered in this book. If you are still hungry for more, this book shows you how to do SDN with OpenFlow.

What This Book Covers

Chapter 1, Introducing OpenFlow, introduces the OpenFlow and its role in the SDN ecosystem and how it works in a computer network. This chapter shapes the required knowledge prior to the actual setup of an experimental environment. The notion of flow, flow forwarding, OpenFlow functions, what can OpenFlow tables do, and features and limitations of OpenFlow are covered in this chapter.

Chapter 2, Implementing the OpenFlow Switch, covers the available implementations of OpenFlow switches including hardware and software implementations.

Chapter 3, The OpenFlow Controllers, covers the role of OpenFlow controllers as a control entity for OpenFlow switches and the provided API (that is, northbound interface) for the development of OpenFlow-based Network Applications (Net Apps).

For More Information:
Chapter 4, Setting Up the Environment, introduces the options for OpenFlow switches and controllers. It also covers the environment for Net App development. This chapter focuses on the installation of virtual machines (VMs) and tools (for example, Mininet and Wireshark), which will be used in the next chapters for Net App development.

Chapter 5, "Net App" Development, covers developing of sample network applications (for example, learning switch and firewall) to show how OpenFlow provides the common ground for network application (Net App) development.

Chapter 6, Getting a Network Slice, covers the network slicing using OpenFlow and FlowVisor. A setup will be planned and the reader can understand how to configure and use a slice of the network using FlowVisor.

Chapter 7, OpenFlow in Cloud Computing, focuses on the role of OpenFlow in cloud computing and in particular, the installation and configuration of OpenStack's Neutron will be covered. Neutron is an incubated OpenStack project that provides network connectivity as a service (NaaS) between interface devices (for example, vNICs or virtual network interface cards), which are managed by other OpenStack services.

Chapter 8, Open Source Resources, explains and gives pointers to the important open source projects that network engineers and/or administrators can utilize in their production environment. These projects range from OpenFlow soft switches, Controllers, virtualization tools, Orchestration tools, to simulation and testing utilities.

Introducing OpenFlow

In order to understand the role of OpenFlow and its building blocks, and how it can be used for OpenFlow-based network application development, it is important to provide a brief introduction of OpenFlow and how it works. This chapter shapes the required knowledge prior to the actual setup of SDN/OpenFlow-enabled experimental and development environment. OpenFlow can be considered as one of the early implementations of the SDN concept. Therefore, before going through OpenFlow, it is worth giving a brief introduction to the SDN and the related activities around it.

Understanding Software Defined Networking – OpenFlow flavor

Software Defined Networking (SDN), often referred to as a revolutionary new idea in computer networking, promises to dramatically simplify network control, management, and enable innovation through network programmability. Computer networks are typically constructed from a large number of network devices (such as switches, routers, firewalls, and so on) with many complex protocols (software), which are implemented and embedded on them. Network engineers are responsible for configuring policies to respond to a wide range of network events and application scenarios. They manually transform these high-level policies into low-level configuration commands. These very complex tasks are often accomplished with access to very limited tools. Thus, network management control and performance tuning are quite challenging and error-prone tasks.

Another challenge is what network engineers and researchers refer to as Internet ossification. Due to its huge deployment base and its impacts on different aspects of our life, the Internet has become extremely difficult to evolve both in terms of its physical infrastructure, along with its protocols and performance. As emerging and demanding applications become more complex, the current status quo of the Internet seems not to be able to evolve to address emerging challenges.

For More Information:
Introducing OpenFlow

The concept of programmable networks has been proposed as a way to facilitate network evolution. In particular, SDN is a new networking paradigm, in which the forwarding hardware (for example, specialized packet forwarding engines) is decoupled from the control decisions (for example, the protocols and control software). The migration of control logic, which used to be tightly integrated in the networking devices (for example, Ethernet switches) into accessible and logically centralized controllers, enables the underlying networking infrastructure to be abstracted from the application’s point of view. This separation provides a more flexible, programmable, vendor-agnostic, cost efficient, and innovative network architecture. Besides the network abstraction, the SDN architecture will provide a set of Application Programming Interfaces (APIs) that simplifies the implementation of common network services (for example, routing, multicast, security, access control, bandwidth management, traffic engineering, QoS, energy efficiency, and various forms of policy management). In SDN, the network intelligence is logically centralized in software-based controllers (at the control plane), and network devices become the simple packet forwarding devices (the data plane) that can be programmed via an open interface. One of the early implementations of this open interface is called OpenFlow.

The separation of the forwarding hardware from the control logic allows easier deployment of new protocols and applications, straightforward network visualization and management, and consolidation of various middle boxes into software control. Instead of enforcing policies and running protocols on a convolution of scattered devices, the network is reduced to simple forwarding hardware and the decision-making network controller(s). The forwarding hardware consists of the following:

1. A flow table containing flow entries consisting of match rules and actions that take on active flows.
2. A transport layer protocol that securely communicates with a controller about new entries that are not currently in the flow table.

Activities around SDN/OpenFlow

While OpenFlow has received a considerable amount of industry attention, it is worth mentioning that the idea of programmable networks and decoupled control plane (control logic) from data plane has been around for many years. The Open Signaling Working Group (OPENSIG) initiated a series of workshops in 1995 to make ATM, Internet, and mobile networks more open, extensible, and programmable. Motivated by these ideas, an Internet Engineering Task Force (IETF) working group came up with General Switch Management Protocol (GSMP), to control a label switch. This group is officially concluded and GSMPv3 was published in June, 2002. The Active

For More Information:
Network initiative proposed the idea of a network infrastructure that would be programmable for customized services. However, Active Network never gathered critical mass, mainly due to practical security and performance concerns. Starting in 2004, the 4D project (www.cs.cmu.edu/~4D/) advocated a clean slate design that emphasized separation between the routing decision logic and the protocols governing the interaction between network elements. The ideas in the 4D project provided direct inspiration for later works such as NOX (www.noxrepo.org), which proposed an operating system for networks in the context of an OpenFlow-enabled network. Later on in 2006, the IETF Network Configuration Protocol working group proposed NETCONF as a management protocol for modifying the configuration of network devices. The working group is currently active and the latest proposed standard was published in June, 2011. The IETF Forwarding and Control Element Separation (ForCES) working group is leading a parallel approach to SDN. SDN and Open Networking Foundation share some common goals with ForCES. With ForCES, the internal network device architecture is redefined as the control element is separated from the forwarding element, but the combined entity is still represented as a single network element to the outside world. The immediate predecessor to OpenFlow was the Stanford's SANE/Ethane project (yuba.stanford.edu/sane, and yuba.stanford.edu/ethane/), which, in 2006, defined a new network architecture for enterprise networks. Ethane's focus was on using a centralized controller to manage policy and security in a network.

A group of network operators, service providers, and vendors have recently created the Open Networking Foundation (www.opennetworking.org), an industrial driven organization, to promote SDN and standardize the OpenFlow protocol. At the time of writing this, the latest specification of OpenFlow was version 1.4. However, since the widely implemented and deployed specification is OpenFlow 1.0.0 (Wire Protocol 0x01), we will limit ourselves to the OpenFlow 1.0.0 in this book.

Building Blocks
The SDN switch (for instance, an OpenFlow switch), the SDN controller, and the interfaces present on the controller for communication with forwarding devices, generally southbound interface (OpenFlow) and network applications interface (northbound interface) are the fundamental building blocks of an SDN deployment. Switches in an SDN are often represented as basic forwarding hardware accessible via an open interface, as the control logic and algorithms are offloaded to a controller. OpenFlow switches come in two varieties: pure (OpenFlow-only) and hybrid (OpenFlow-enabled).
Introducing OpenFlow

Pure OpenFlow switches have no legacy features or on-board control, and completely rely on a controller for forwarding decisions. Hybrid switches support OpenFlow in addition to traditional operation and protocols. Most commercial switches available today are hybrids. An OpenFlow switch consists of a flow table, which performs packet lookup and forwarding. Each flow table in the switch holds a set of flow entries that consists of:

1. Header fields or match fields, with information found in packet header, ingress port, and metadata, used to match incoming packets.
2. Counters, used to collect statistics for the particular flow, such as number of received packets, number of bytes, and duration of the flow.
3. A set of instructions or actions to be applied after a match that dictates how to handle matching packets. For instance, the action might be to forward a packet out to a specified port.

The decoupled system in SDN (and OpenFlow) can be compared to an application program and an operating system in a computing platform. In SDN, the controller (that is, network operating system) provides a programmatic interface to the network, where applications can be written to perform control and management tasks, and offer new functionalities. A layered view of this model is illustrated in the following figure. This view assumes that the control is centralized and applications are written as if the network is a single system. While this simplifies policy enforcement and management tasks, the bindings must be closely maintained between the control and the network forwarding elements. As shown in the following figure, a controller that strives to act as a network operating system must implement at least two interfaces: a southbound interface (for example, OpenFlow) that allows switches to communicate with the controller and a northbound interface that presents a programmable API to network control and high-level policy applications/services. Header fields (match fields) are shown in the following figure. Each entry of the flow table contains a specific value, or ANY (* or wildcard, as depicted in the following figure), which matches any value.

For More Information:
OpenFlow switch, Flow table, OpenFlow controller, and network applications.

If the switch supports subnet masks on the IP source and/or destination fields, these can more precisely specify matches. The port field (or ingress port) numerically represents the incoming port of the switch and starts at 1. The length of this field is implementation dependent. The ingress port field is applicable to all packets. The source and destination MAC (Ethernet) addresses are applicable to all packets on enabled ports of the switch and their length is 48 bits. The Ethernet type field is 16 bits wide and is applicable to all the packets on enabled ports. An OpenFlow switch must match the type in both standard Ethernet and IEEE 802.2 with a Subnetwork Access Protocol (SNAP) header and Organizationally Unique Identifier (OUI) of 0x000000. The special value of 0x05FF is used to match all the 802.3 packets without SNAP headers. VLAN ID is applicable to all packets with an Ethernet type of 0x8100. The size of this field is 12 bits (that is, 4096 VLANs). The VLAN priority (or the VLAN PCP field) is 3 bits wide and is applicable to all packets of Ethernet type 0x8100. The IP source and destination address fields are 32 bit entities and are applicable to all IP and ARP packets. These fields can be masked with a subnet mask. The IP protocol field is applicable to all IP, IP over Ethernet, and the ARP packets. Its length is 8 bits and in case of ARP packets, only the lower 8 bits of the ARP op-code are used. The IP ToS (Type of Service) bits has a length of 6 bits and is applicable to all IP packets. It specifies an 8 bit value and places ToS in the upper 6 bits. The source and destination transport port addresses (or ICMP type/code) have a length of 16 bits and are applicable to all TCP, UDP, and ICMP packets. In case of the ICMP type/code only the lower 8 bits are considered for matching.

For More Information:
Counters are maintained per table, per flow, per port and per queue. Counters wrap around with no overflow indicator. The required set of counters is summarized in the following figure. The phrase byte in this figure (and throughout this book) refers to an 8 bit octet. Duration refers to the time the flow has been installed in the flow table of the switch. The receive errors field includes all explicitly specified errors, including frame, overrun, and CRC errors, plus any others.

<table>
<thead>
<tr>
<th>Per Port Counters:</th>
<th>Per Table Counters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Received Packets (64 bits)</td>
<td>Active Entries (32 bits)</td>
</tr>
<tr>
<td>Transmitted Packets (64 bits)</td>
<td>Packet lookups (64 bits)</td>
</tr>
<tr>
<td>Received Bytes (64 bits)</td>
<td>Packet Matches (64 bits)</td>
</tr>
<tr>
<td>Transmitted Bytes (64 bits)</td>
<td></td>
</tr>
<tr>
<td>Receive Drops (64 bits)</td>
<td></td>
</tr>
<tr>
<td>Transmit Drops (64 bits)</td>
<td></td>
</tr>
<tr>
<td>Receive Errors (64 bits)</td>
<td></td>
</tr>
<tr>
<td>Transmit Errors (64 bits)</td>
<td></td>
</tr>
<tr>
<td>Receive Frame Alignment Errors (64 bits)</td>
<td></td>
</tr>
<tr>
<td>Receive Overrun Errors (64 bits)</td>
<td></td>
</tr>
<tr>
<td>Receive CRC Errors (64 bits)</td>
<td></td>
</tr>
<tr>
<td>Collisions (64 bits)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Per Flow Counters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Received Packets (64 bits)</td>
</tr>
<tr>
<td>Received Bytes (64 bits)</td>
</tr>
<tr>
<td>Transmit Drop (64 bits)</td>
</tr>
<tr>
<td>Duration (seconds) (32 bits)</td>
</tr>
<tr>
<td>Duration (nano seconds) (32 bits)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Per Queue Counters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmitted Packets (64 bits)</td>
</tr>
<tr>
<td>Transmitted Bytes (64 bits)</td>
</tr>
<tr>
<td>Transmit Overrun Errors (64 bits)</td>
</tr>
</tbody>
</table>

Required list of counters for use in statistical messages.

Each flow entry is associated with zero or more actions that instruct the OpenFlow switch how to handle matching packets. If no forward actions are present, the packet is dropped. Action lists must be processed in the specified order. However, there is no guaranteed packet output ordering within an individual port. For instance, two packets which are generated and destined to a single output port as part of the action processing, may be arbitrarily re-ordered. Pure OpenFlow switches only support the Required Actions, while hybrid OpenFlow switches may also support the NORMAL action. Either type of switches can also support the FLOOD action. The Required Actions are:

- **Forward**: OpenFlow switches must support forwarding the packet to physical ports and the following virtual ones:
  - **ALL**: Send the packet on to all interfaces, excluding the incoming port
  - **CONTROLLER**: Encapsulate and send the packet to the controller
  - **LOCAL**: Send the packet to the local networking stack of the switch
  - **TABLE** (Only for packet-out message): Perform action in the flow table
  - **IN_PORT**: Send the packet out the input port

For More Information:

• **Drop**: This indicates that all the matching packets should be dropped. A flow entry with no specified action is considered as a Drop action.

• The **Optional Actions** are:
  - **Forward**: A switch may optionally support the following virtual ports for forward action:
    - **NORMAL**: Process the packet using the traditional forwarding path supported by the switch (that is traditional L2, VLAN, and/or L3 processing)
    - **FLOOD**: Flood the packet along the minimum spanning tree, not including the incoming interface.

• **Enqueue**: This forwards a packet through a queue attached to a port. Forwarding behavior is dictated by the configuration of the queue and is used to provide the basic QoS support.

• **Modify field**: The optional (recommended) field modification actions are:
  - Setting VLAN ID: If no VLAN is present, a new header is added with the specified VLAN ID (12 bit associated data) and priority of zero. If a VLAN header already exists, the VLAN ID is replaced with the specified value.
  - Setting VLAN priority: If no VLAN is present, a new header is added with the specified priority (3 bit associated data) and VLAN ID of zero. If a VLAN ID header already exists, the priority field is replaced with the specified value.
  - Striping the VLAN header: This Strip VLAN header if present.
  - Modifying the Ethernet source/destination MAC address: This replaces the existing Ethernet source/destination MAC address with the new value (specified as a 48 bits data).
  - Modifying the IPv4 source/destination address: This replaces the existing IP source/destination address with a new value (associated with a 32 bits data) and updates the IP checksum (and TCP/UDP checksum if applicable). This action is only applicable to IPv4 packets.
  - Modifying the IPv4 ToS bits: This replace the existing IP ToS field with the 6 bits associated data. This action is only applicable to IPv4 packets.
  - Modifying the transport source/destination port: This replaces the existing TCP/UDP source/destination port with associated 16 bits data and updates the TCP/UDP checksum. This action is only applicable to TCP and UDP packets.

---

**For More Information:**
Upon a packet arrival at the OpenFlow switch, the packet header fields are extracted and matched against the matching fields' portion of the flow table entries. This matching starts at the first flow table entry and continues through subsequent flow table entries. If a matching entry is found, the switch applies the appropriate set of instructions associated with the matched flow entry. For each packet that matches a flow entry, the associated counters for that entry are updated. If the flow table look-up procedure does not result on a match, the action taken by the switch will depend on the instructions defined at the table-miss flow entry. The flow table must contain a table-miss entry in order to handle table misses. This particular entry specifies a set of actions to be performed when no match is found for an incoming packet. These actions include dropping the packet, sending the packet out on all interfaces, or forwarding the packet to the controller over the secure OpenFlow channel. Header fields used for the table lookup depend on the packet types as described below:

- Rules specifying a port (ingress port) are matched against the physical port that received the packet.
- The Ethernet headers (Source MAC, Destination MAC, Ethernet Type, and more) as specified in the first figure, and are used for all packets.
- If the packet is a VLAN (Ethernet type 0x8100), the VLAN ID and VLAN priority (PCP) fields are used in the lookup.
- For IP packets (Ethernet type equal to 0x0800), the lookup fields also include those in the IP header (Source IP, Destination IP, IP protocol, ToS, and so on).
- For IP packets that are TCP or UDP (IP protocol equal to 6 or 17), the lookup includes the transport ports (TCP/UDP source/destination ports).
- For IP packets that are ICMP (IP protocol equal to 1), the lookup includes the Type and Code fields.
- For IP packets with nonzero fragment offset or more fragments bit set, the transport ports are set to zero for the lookup.
- Optionally, for ARP packets (Ethernet type equal to 0x0806), the lookup fields may also include the contained IP source and destination fields.

Packets are matched against flow entries based on prioritization. An entry that specifies an exact match (that is no wildcards) is always the highest priority. All wildcard entries have a priority associated with them. Higher priority entries must match before lower priority ones. If multiple entries have the same priority, the switch is free to choose any ordering. Higher numbers have higher priorities. The following figure shows the packet flow in an OpenFlow switch. It is important to note that if a flow table field has a value of ANY (*, wildcard), it matches all the possible values in the header.
There are various Ethernet framing types (Ethernet II, 802.3 with or without SNAP, and so on). If the packet is an Ethernet II frame, the Ethernet type is handled in the expected way. If the packet is an 802.3 frame with a SNAP header and an OUI equal to \texttt{0x000000}, the SNAP protocol ID is matched against the flow’s Ethernet type. A flow entry that specified an Ethernet Type of \texttt{0x05FF}, matches all Ethernet 802.2 frames without a SNAP header and those with SNAP headers that do not have an OUI of \texttt{0x000000}.

### Packet flow in an OpenFlow switch.

#### OpenFlow messages

The communication between the controller and switch happens using the OpenFlow protocol, where a set of defined messages can be exchanged between these entities over a secure channel. The secure channel is the interface that connects each OpenFlow switch to a controller. The \textbf{Transport Layer Security (TLS)} connection to the user-defined (otherwise fixed) controller is initiated by the switch on its power on. The controller’s default TCP port is \texttt{6633}. The switch and controller mutually authenticate by exchanging certificates signed by a site-specific private key. Each switch must be user-configurable with one certificate for authenticating the controller (controller certificate) and the other for authenticating to the controller (switch certificate).

---

**For More Information:**

Introducing OpenFlow

Traffic to and from the secure channel is not checked against the flow table and therefore the switch must identify incoming traffic as local before checking it against the flow table. In the case that a switch loses contact with the controller, as a result of an echo request timeout, TLS session timeout, or other disconnection, it should attempt to contact one or more backup controllers. If some number of attempts to contact a controller (zero or more) fail, the switch must enter emergency mode and immediately reset the current TCP connection. Then the matching process is dictated by the emergency flow table entries (marked with the emergency bit set). Emergency flow modify messages must have timeout value set to zero. Otherwise, the switch must refuse the addition and respond with an error message. All normal entries are deleted when entering emergency mode. Upon connecting to a controller again, the emergency flow entries remain. The controller then has the option of deleting all the flow entries, if desired.

The first time a switch boots up, it is considered to be in emergency mode. Configuration of the default set of flow entries is outside the scope of the OpenFlow protocol.

The controller configures and manages the switch, receives events from the switch, and sends packets out to the switch through this interface. Using the OpenFlow protocol, a remote controller can add, update, or delete flow entries from the switch's flow table. That can happen reactively (in response to a packet arrival) or proactively. The OpenFlow protocol can be viewed as one possible implementation of controller-switch interactions (southbound interface), as it defines the communication between the switching hardware and a network controller. For security, OpenFlow 1.3.x provides optional support for encrypted TLS communication and a certificate exchange between the switches/controller(s); however, the exact implementation and certificate format is not currently specified. Also, fine-grained security options regarding scenarios with multiple controllers are outside the scope of the current specification, as there is no specific method to only grant partial access permissions to an authorized controller. The OpenFlow protocol defines three message types, each with multiple subtypes:

- Controller-to-switch
- Symmetric
- Asynchronous

Controller-to-switch

Controller-to-switch messages are initiated by the controller and used to directly manage or inspect the state of the switch. This type of messages may or may not require a response from the switch and are categorized in the following subtypes.

For More Information:
Features
Upon establishment of the TLS session, the controller sends a feature request message to the switch. The switch must reply with a features reply message that specifies the features and capabilities that are supported by the switch.

Configuration
The controller is able to set and query configuration parameters in the switch. The switch only responds to a query from the controller.

Modify-State
These messages are sent by the controller to manage the state of the switches. They are used to add/delete or modify flow table entries or to set switch port priorities. Flow table modification messages can have the following types:

• **ADD**: For the ADD requests with the OFPFF_CHECK_OVERLAP flag set, the switch must first check for any overlapping flow entries. Two flow entries overlap if a single packet may match both, and both entries have the same priority. If an overlap conflict exists between an existing flow entry and the ADD request, the switch must refuse the addition and respond with ofp_error_msg with the OFPET_FLOW_MODE_FAILED error type and the OFPFOFMFC_OVERLAP error code. For the valid (non-overlapping) ADD requests, or those with no overlap checking flag set, the switch must insert the flow entry at the lowest numbered table entry for which the switch supports all wildcards set in the flow_match struct, and for which the priority would be observed during the matching process. If a flow entry with identical header fields and priority already resides in the flow table, then that entry including its counters must be removed and the new flow entry must be added. If a switch cannot find any table entry to add the incoming flow entry, the switch should send ofp_error_msg with the OFPET_FLOW_MOD_FAILED type and the OFPFOFMFC_ALL_TABLES_FULL error code. If the action list in a flow modify message references a port that will never be valid on a switch, the switch must return ofp_error_msg with the OFPET_BAD_ACTION type and the OFPFOFBAD_BAD_OUT code. If the referenced port may be valid in the future (for example, when a line card is added to a chassis) the switch may either silently drop packets sent to the referenced port, or immediately return an OFPFOFBAD_BAD_PORT error and refuse the flow modify message.
• **MODIFY**: If a flow entry with an identical header field does not currently reside in the flow table, the **MODIFY** command acts like an **ADD** command, and the new flow entry must be inserted with zeroed counters. Otherwise the actions field is changed on the existing entry and its counters and idle timeout fields are left unchanged.

• **DELETE**: For delete requests, if no flow entry matches, no error is recorded and no flow table modification occurs. If a flow entry matches, the entry must be deleted, and then each normal entry with the `OFPFF_SEND_FLOW_REM` flag set should generate a flow removal message. Deleted emergency flow entries generate no flow removal messages. **DELETE** and **DELETE STRICT** (see next bullet point) commands can be optionally filtered by the output port. If the `out_port` field contains a value other than `OFPP_NONE`, it introduces a constraint when matching. This constraint is that the rule must contain an output action directed at that port. This field is ignored by the **ADD**, **MODIFY**, and **MODIFY STRICT** messages.

• **MODIFY** and **DELETE**: These flow mode commands have corresponding **_STRICT** versions. In non-RESTRICT versions, the wildcards are active and all flows that match the description are modified or removed. In **_STRICT** versions, all fields, including the wildcards and priority, are strictly matched against the entry and only an identical flow is modified or removed. For instance, if a message to remove entries is sent to the switch that has all wildcard flags set, the **DELETE** command would delete all flows from all tables. However, the **DELETE STRICT** command would only delete a rule that applies to all packets at the specified priority. For the non-strict **MODIFY** and **DELETE** commands that contain wildcards, a match will occur when a flow entry exactly matches or is more specific than the description in the **flow_mod** command. For example, if a **DELETE** command says to delete all flows with a destination port of 80, then a flow entry that has all wildcards will not be deleted. However, a **DELETE** command that has all wildcards will delete an entry that matches all port 80 traffic.

**Read-State**
These messages collect statistics from the switch flow tables, ports, and the individual flow entries.

**Send-Packet**
These are used by the controller to send packets out of a specified port on the switch.

---

For More Information:
Barrier
Barrier request/reply messages are used by the controller to ensure message dependencies have been met or to receive notifications for completed operations.

Symmetric messages
Symmetric messages are initiated by either the switch or the controller and sent without solicitation. There are three symmetric message subtypes in OpenFlow protocol as follows:

Hello
Hello messages are exchanged between the switch and controller upon connection setup.

Echo
Echo request/reply messages can be sent from either the switch or the controller, and must return an echo reply. These messages can be used to indicate the latency, bandwidth, and/or liveliness of a controller-switch connection (that is a heartbeat).

Vendor
These messages provide a standard way for OpenFlow switches to offer additional functionality within the OpenFlow message type space for future revisions of OpenFlow.

Asynchronous messages
Asynchronous messages are initiated by the switch and used to update the controller of network events and changes to the switch state. Switches send asynchronous messages to the controller to denote a packet arrival, switch state change, or an error. There are four main asynchronous messages as follows:

For More Information:
Introducing OpenFlow

Packet-in
For all packets that do not have a matching flow entry or if a packet matches an entry with a send to controller action, a packet-in message is sent to the controller. If the switch has sufficient memory to buffer packets that are sent to the controller, the packet-in message contains some fraction of the packet header (by default, 128 bytes) and a buffer ID to be used by the controller when it is ready for the switch to forward the packet. Switches that do not support internal buffering (or have run out of internal buffer space) must send the full packet to the controller as part of the message.

Flow-Removal
When a flow entry is added to the switch by a flow modify message (the Modify State section), an idle timeout value indicates when the entry should be removed due to the lack of activity as well as a hard timeout value. The hard timeout value indicates when the entry should be removed, regardless of activity. The flow modify message also specifies whether the switch should send a flow removal message to the controller when the flow expires. Flow modify messages, which delete flow entries may also cause flow removal messages.

Port-status
The switch is expected to send port-status messages to the controller as the port configuration state changes. These events include changes in port status (for example, disabled by the user) or a change in the port status as specified by 802.1D (Spanning Tree). OpenFlow switches may optionally support 802.1D Spanning Tree Protocol (STP). These switches are expected to process all 802.1D packets locally before performing flow lookup. Ports status as specified by the STP is then used to limit packets forwarded to the OFF_FLOOD port to only those ports along the spanning tree. Port changes as a result of the spanning tree are sent to the controller via the port-update messages. Note that forward actions that specify the outgoing port of OFF_ALL ignore the port status set by the STP. Packets received on the ports that are disabled by the STP must follow the normal flow table processing path.

Error
The switch is able to notify the controller of problems using error messages.

The heart of OpenFlow specification is the set of C structures used for OpenFlow protocol messages. Interested readers can find these data structures and their detailed explanation available at:
www.openflow.org/documents/openflow-spec-v1.0.0.pdf or
www.opennetworking.org/sdn-resources/onf-specifications.

For More Information:
Northbound interface

External management systems or network applications (Net Apps) may wish to extract information about the underlying network or control an aspect of the network behavior or policy. Additionally, controllers may find it necessary to communicate with each other for a variety of reasons. For instance, an internal control application may need to reserve resources across multiple domains of control, or a primary controller may need to share policy information with a backup controller. Unlike controller-switch communication (that is the southbound interface), there is no currently accepted standard for northbound interface and they are more likely to be implemented on an ad-hoc basis for particular applications. One potential reason is that the northbound interface is defined entirely in software, while controller-switch interactions must enable the hardware implementation. If we consider the controller as a network operating system, then there should be a clearly defined interface by which applications can access the underlying hardware (switches), coexist and interact with other applications, and utilize system services (for example, topology discovery, forwarding, and so on), without requiring the application developer to know the implementation details of the controller (that is the network operating system). While there are several controllers that exist, their application interfaces are still in the early stages and independent from each other and incompatible. Until a clear northbound interface standard emerges, SDN applications will continue to be developed in an ad-hoc fashion and the concept of flexible and portable network apps may have to wait for some time.

Summary

The OpenFlow is the continuation of many previous efforts to provide decoupled control and data forwarding in networking equipment. A background of these efforts was presented in this chapter. Presenting the key building blocks of an SDN deployment, in particular the OpenFlow protocol and its basic operation were covered in this chapter. After introducing OpenFlow, in the next chapter we present the reference implementation of OpenFlow switch in software and hardware along with an introduction to Mininet experiment environment.

For More Information:

Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.