Chapter 4. SDN Data Plane and OpenFlow

> "I tell you," went on Syme with passion, "that every time a train comes in I feel that it has broken past batteries of besiegers, and that man has won a battle against chaos. You say contemptuously that when one has left Sloane Square one must come to Victoria. I say that one might do a thousand things instead, and that whenever I really come there I have the sense of hairbreadth escape. And when I hear the guard shout out the word 'Victoria', it is not an unmeaning word. It is to me the cry of a herald announcing conquest. It is to me indeed 'Victoria'; it is the victory of Adam."
>
> —*The Man Who Was Thursday*, G. K. Chesterton

Chapter Objectives: **After studying this chapter, you should be able to**

- Present an overview of the functions of the SDN data plane.
- Understand the concept of an OpenFlow logical network device.
- Describe and explain the OpenFlow flow table entry structure.
- Summarize the operation of the OpenFlow pipeline.
- Explain the operation of the group table.
- Understand the basic elements of the OpenFlow protocol.

Section 4.1 of this chapter begins the detailed study of software-defined networking (SDN) with a discussion of the data plane (Figure 4.1). The remainder of the chapter is devoted to OpenFlow, the most widely used implementation of the SDN data plane. OpenFlow is both a specification of the logical structure of data plane functionality and a protocol between SDN controllers and network devices. Sections 4.2 and 4.3, respectively, examine the OpenFlow logical network device and the OpenFlow protocol in more detail.

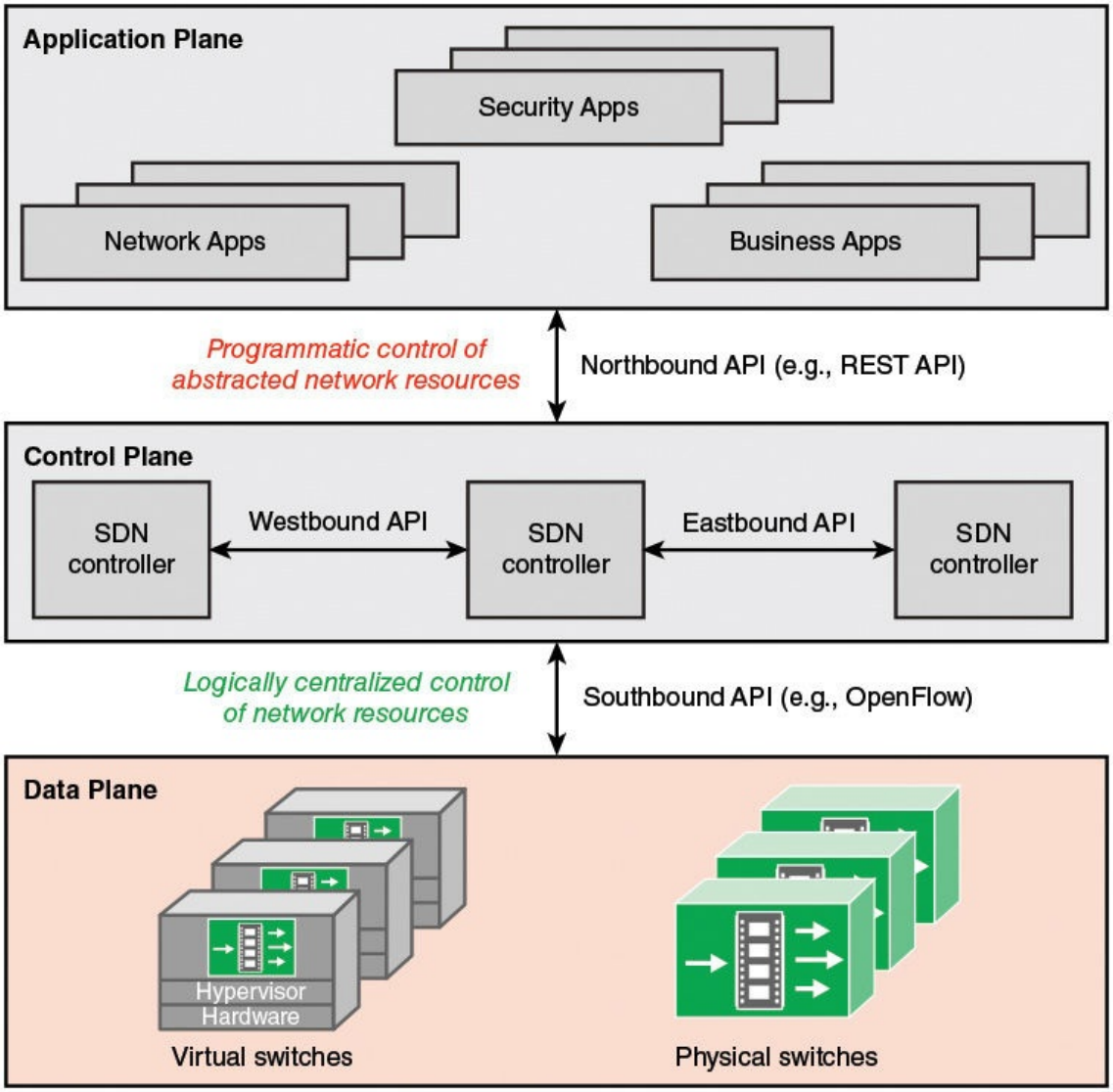

FIGURE 4.1 SDN Architecture

4.1 SDN Data Plane

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

Data Plane Functions

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

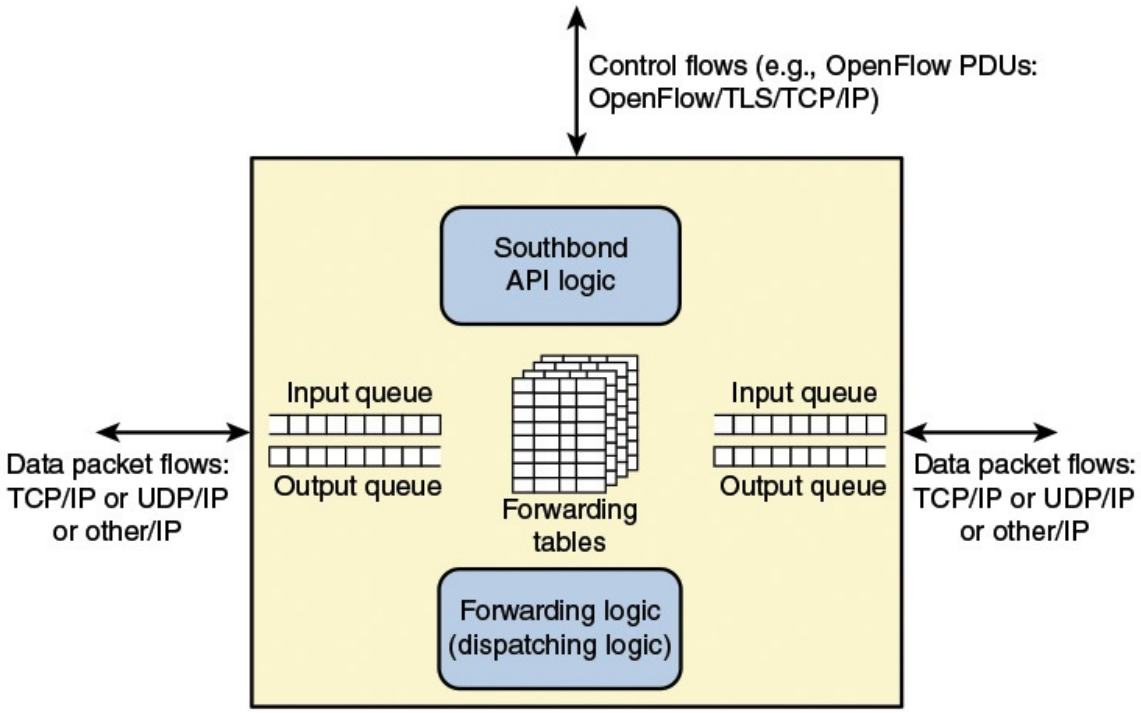

FIGURE 4.2 Data Plane Network Device

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

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

arriving packets may be placed in an input queue, awaiting processing by the network device, and forwarded packets are generally placed in an output queue, awaiting transmission.

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

Data Plane Protocols

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

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

4.2 OpenFlow Logical Network Device

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

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

These requirements are addressed by OpenFlow, which is both a protocol between SDN controllers and network devices and a specification of the logical structure of the network switch functionality. OpenFlow is defined in the

OpenFlow Switch Specification, published by the Open Networking Foundation (ONF).

Open Network Foundation OpenFlow Definition

This section covers the logical switch architecture defined by OpenFlow. Our discussion is based on the OpenFlow specification current at the time of this writing: Version 1.5.1, March 26, 2015.

Figure 4.3 indicates the main elements of an OpenFlow environment, consisting of SDN controllers that include OpenFlow software, OpenFlow switches, and end systems.

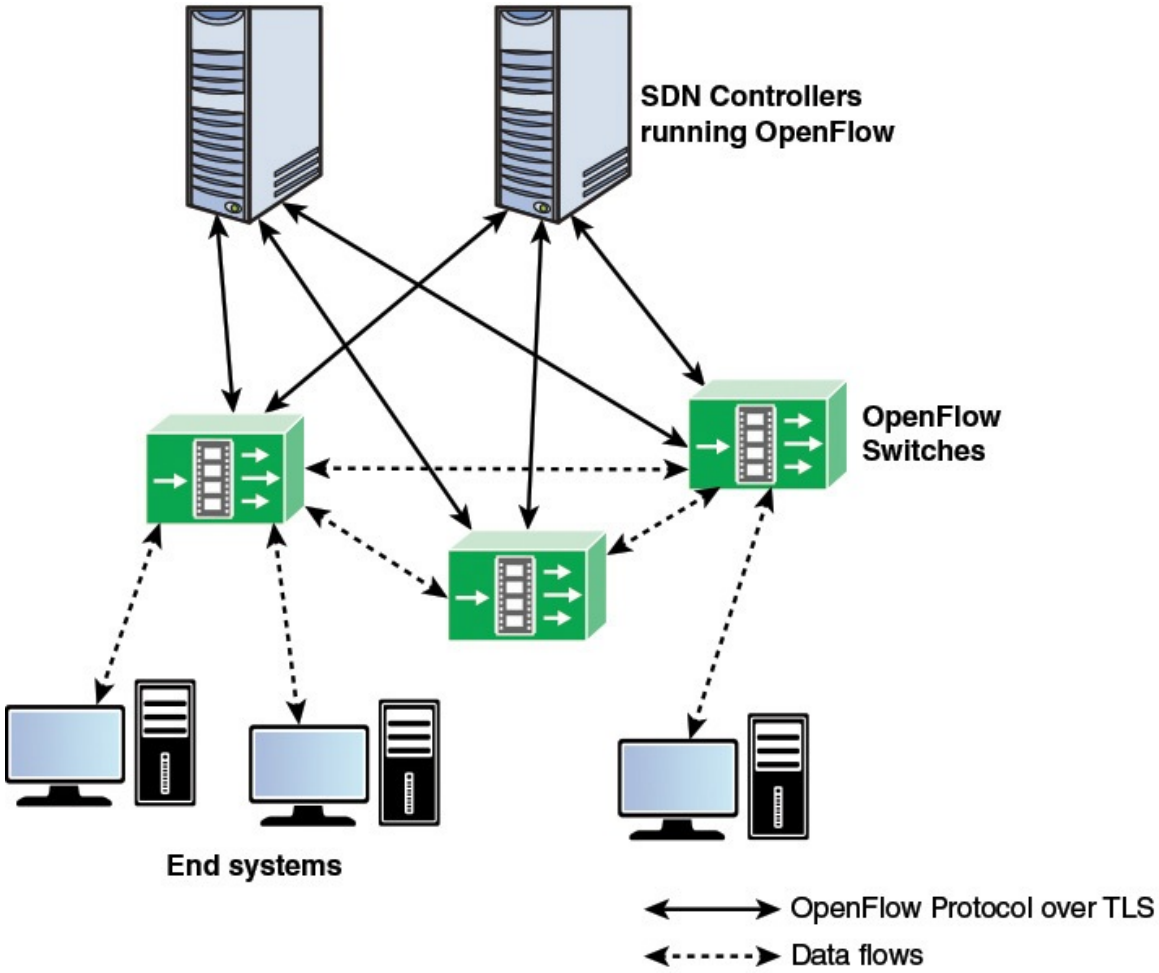

FIGURE 4.3 OpenFlow Switch Context

Figure 4.4 displays the main components of an OpenFlow switch. An SDN controller communicates with OpenFlow-compatible switches using the OpenFlow protocol running over Transport Layer Security (TLS). Each switch connects to other **OpenFlow switches** and, possibly, to end-user devices that are the sources and destinations of packet flows. On the switch side, the interface is known as an **OpenFlow channel**. These connections are via OpenFlow ports. An **OpenFlow port** also connects the switch to the SDN controller. OpenFlow defines three types of ports:

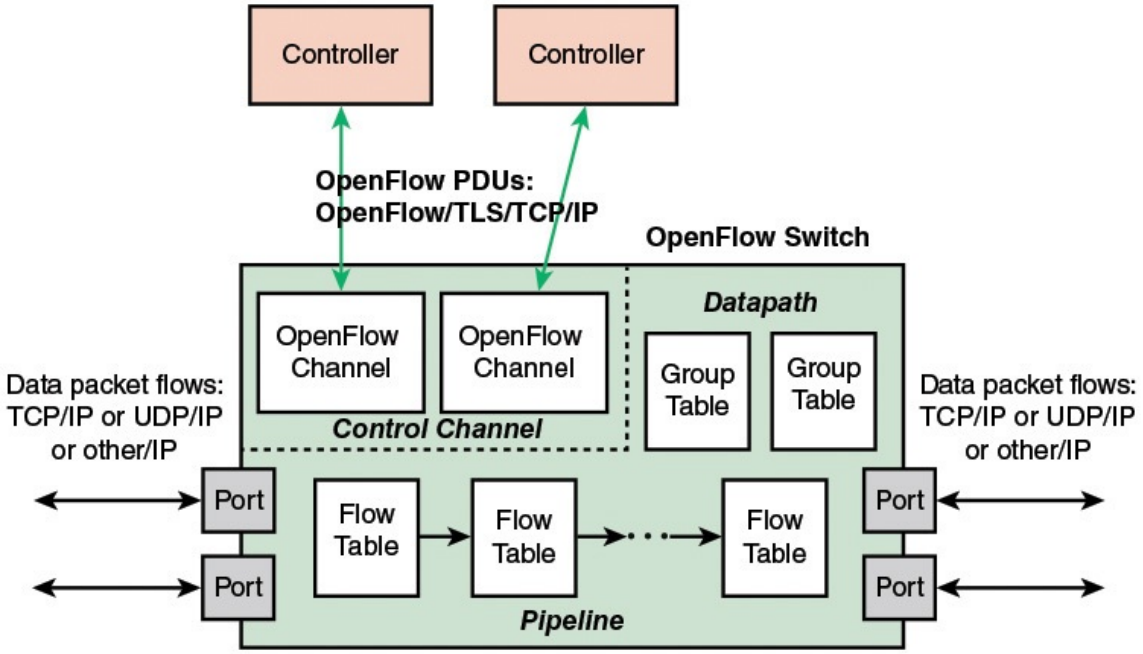

FIGURE 4.4 OpenFlow Switch

- **Physical port:** Corresponds to a hardware interface of the switch. For example, on an Ethernet switch, physical ports map one to one to the Ethernet interfaces.
- **Logical port:** Does not correspond directly to a hardware interface of the switch. Logical ports are higher-level abstractions that may be defined in the switch using non-OpenFlow methods (for example, link aggregation groups, tunnels, loopback interfaces). Logical ports may include packet encapsulation and may map to various physical ports. The processing done by the logical port is implementation dependent and must be transparent to OpenFlow processing, and those ports must interact with OpenFlow processing like OpenFlow physical ports.
- **Reserved port:** Defined by the OpenFlow specification. It specifies generic forwarding actions such as sending to and receiving from the controller, flooding, or forwarding using non-OpenFlow methods, such as “normal” switch processing.

Within each switch, a series of tables is used to manage the flows of packets through the switch.

The OpenFlow specification defines three types of tables in the logical switch architecture. A **flow table** matches incoming packets to a particular flow and

specifies what functions are to be performed on the packets. There may be multiple flow tables that operate in a pipeline fashion, as explained subsequently. A flow table may direct a flow to a **group table**, which may trigger a variety of actions that affect one or more flows. A **meter table** can trigger a variety of performance-related actions on a flow. Meter tables are discussed in Chapter 10. Using the OpenFlow switch protocol, the controller can add, update, and delete flow entries in tables, both reactively (in response to packets) and proactively.

➔ *See Chapter 10, "Quality of Service"*

Before proceeding, it is helpful to define what is meant by the term *flow*. Curiously, this term is not defined in the OpenFlow specification, nor is there an attempt to define it in virtually all of the literature on OpenFlow. In general terms, a flow is a sequence of packets traversing a network that share a set of header field values. For example, a flow could consist of all packets with the same source and destination IP addresses or all packets with the same virtual LAN (VLAN) identifier. The sections that follow provide a more specific definition of this concept.

Flow Table Structure

The basic building block of the logical switch architecture is the **flow table**. Each packet that enters a switch passes through one of more flow tables. Each flow table consists of a number of rows, called **entries**, consisting of seven components (see part a of Figure 4.5), as defined in the list that follows.

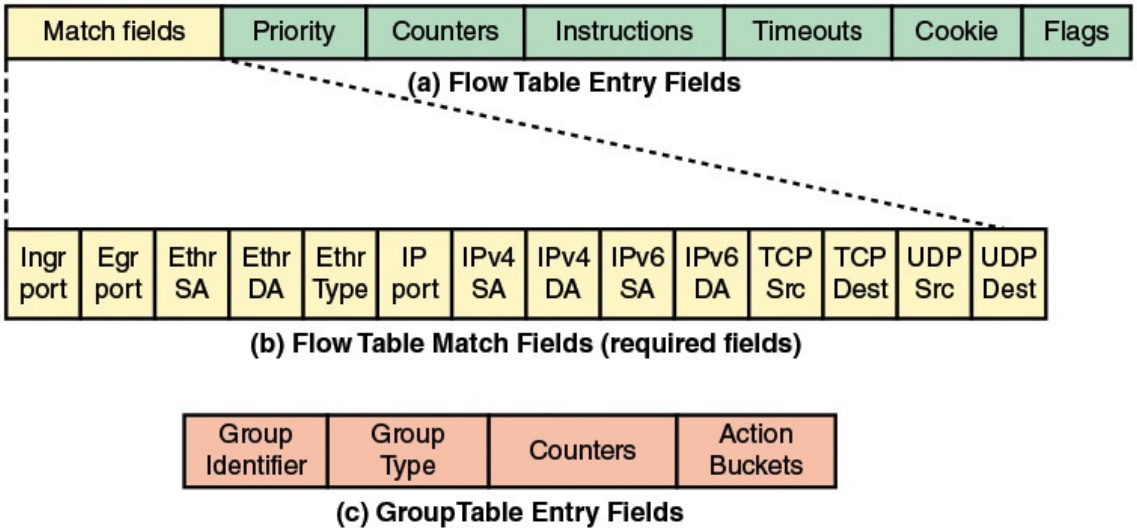

FIGURE 4.5 OpenFlow Table Entry Formats

- **Match fields:** Used to select packets that match the values in the fields.
- **Priority:** Relative priority of table entries. This is a 16-bit field with 0 corresponding to the lowest priority. In principle, there could be 2^{16} = 64k priority levels.
- **Counters:** Updated for matching packets. The OpenFlow specification defines a variety of counters. Table 4.1 lists the counters that must be supported by an OpenFlow switch.

Counter	Usage	Bit Length
Reference count (active entries)	Per flow table	32
Duration (seconds)	Per flow entry	32
Received packets	Per port	64
Transmitted packets	Per port	64
Duration (seconds)	Per port	32
Transmit packets	Per queue	64
Duration (seconds)	Per queue	32
Duration (seconds)	Per group	32
Duration (seconds)	Per meter	32

TABLE 4.1 Required OpenFlow Counters

- **Instructions:** Instructions to be performed if a match occurs.

- **Timeouts:** Maximum amount of idle time before a flow is expired by the switch. Each flow entry has an idle_timeout and a hard_timeout associated with it. A nonzero hard_timeout field causes the flow entry to be removed after the given number of seconds, regardless of how many packets it has matched. A nonzero idle_timeout field causes the flow entry to be removed when it has matched no packets in the given number of seconds.
- **Cookie:** 64-bit opaque data value chosen by the controller. May be used by the controller to filter flow statistics, flow modification and flow deletion; not used when processing packets.
- **Flags**: Flags alter the way flow entries are managed; for example, the flag OFPFF_SEND_FLOW_REM triggers flow removed messages for that flow entry.

Match Fields Component

The match fields component of a table entry consists of the following required fields (see part b of Figure 4.5):

- **Ingress port:** The identifier of the port on this switch on which the packet arrived. This may be a physical port or a switch-defined virtual port. Required in ingress tables.
- **Egress port:** The identifier of the egress port from action set. Required in egress tables.
- **Ethernet source and destination addresses:** Each entry can be an exact address, a bitmasked value for which only some of the address bits are checked, or a wildcard value (match any value).
- **Ethernet type field:** Indicates type of the Ethernet packet payload.
- **IP:** Version 4 or 6.
- **IPv4 or IPv6 source address, and destination address:** Each entry can be an exact address, a bitmasked value, a subnet mask value, or a wildcard value.
- **TCP source and destination ports:** Exact match or wildcard value.
- **UDP source and destination ports:** Exact match or wildcard value.

The preceding match fields must be supported by any OpenFlow-compliant switch. The following fields may be optionally supported.

- **Physical port:** Used to designate underlying physical port when packet is received on a logical port.
- **Metadata:** Additional information that can be passed from one table to another during the processing of a packet. Its use is discussed subsequently.
- **VLAN ID and VLAN user priority:** Fields in the IEEE 802.1Q virtual LAN header. SDN support for VLANs is discussed in Chapter 8, "NFV Functionality."
- **IPv4 or IPv6 DS and ECN:** Differentiated Services and Explicit Congestion Notification fields.
- **SCTP source and destination ports:** Exact match or wildcard value for Stream Transmission Control Protocol.
- **ICMP type and code fields:** Exact match or wildcard value.
- **ARP opcode:** Exact match in Ethernet Type field.
- **Source and target IPv4 addresses in ARP payload:** Can be an exact address, a bitmasked value, a subnet mask value, or a wildcard value.
- **IPv6 flow label:** Exact match or wildcard.
- **ICMPv6 type and code fields:** Exact match or wildcard value.
- **IPv6 neighbor discovery target address:** In an IPv6 Neighbor Discovery message.
- **IPv6 neighbor discovery source and target addresses:** Link-layer address options in an IPv6 Neighbor Discovery message.
- **MPLS label value, traffic class, and BoS:** Fields in the top label of an MPLS label stack.
- **Provider bridge traffic ISID:** Service instance identifier.
- **Tunnel ID:** Metadata associated with a logical port.
- **TCP flags:** Flag bits in the TCP header. May be used to detect start and end of TCP connections.
- **IPv6 extension**: Extension header.

Thus, OpenFlow can be used with network traffic involving a variety of protocols and network services. Note that at the MAC/link layer, only Ethernet is supported. Therefore, OpenFlow as currently defined cannot control Layer 2 traffic over wireless networks.

Each of the fields in the match fields component either has a specific value or a wildcard value, which matches any value in the corresponding packet header field. A flow table may include a table-miss flow entry, which wildcards all match fields (every field is a match regardless of value) and has the lowest priority.

We can now offer a more precise definition of the term *flow*. From the point of view of an individual switch, a flow is a sequence of packets that matches a specific entry in a flow table. The definition is packet oriented, in the sense that it is a function of the values of header fields of the packets that constitute the flow, and not a function of the path they follow through the network. A combination of flow entries on multiple switches defines a flow that is bound to a specific path.

Instructions Component

The instructions component of a table entry consists of a set of instructions that are executed if the packet matches the entry. Before describing the types of instructions, we need to define the terms *action* and *action set*. Actions describe packet forwarding, packet modification, and group table processing operations. The OpenFlow specification includes the following actions:

- **Output:** Forward packet to specified port. The port could be an output port to another switch or the port to the controller. In the latter case, the packet is encapsulated in a message to the controller.
- **Set-Queue:** Sets the queue ID for a packet. When the packet is forwarded to a port using the output action, the queue ID determines which queue attached to this port is used for scheduling and forwarding the packet. Forwarding behavior is dictated by the configuration of the queue and is used to provide basic QoS support. SDN support for QoS is discussed in Chapter 10.
- **Group:** Process packet through specified group.
- **Push-Tag/Pop-Tag:** Push or pop a tag field for a VLAN or Multiprotocol

Label Switching (MPLS) packet.

- **Set-Field:** The various Set-Field actions are identified by their field type and modify the values of respective header fields in the packet.
- **Change-TTL:** The various Change-TTL actions modify the values of the IPv4 TTL (time to live), IPv6 hop limit, or MPLS TTL in the packet.
- **Drop**: There is no explicit action to represent drops. Instead, packets whose action sets have no output action should be dropped.

An action set is a list of actions associated with a packet that are accumulated while the packet is processed by each table and that are executed when the packet exits the processing pipeline.

The types of instructions can be grouped into four categories:

- **Direct packet through pipeline:** The Goto-Table instruction directs the packet to a table farther along in the pipeline. The Meter instruction directs the packet to a specified meter.
- **Perform action on packet:** Actions may be performed on the packet when it is matched to a table entry. The Apply-Actions instruction applies the specified actions immediately, without any change to the action set associated with this packet. This instruction may be used to modify the packet between two tables in the pipeline.
- **Update action set:** The Write-Actions instruction merges specified actions into the current action set for this packet. The Clear-Actions instruction clears all the actions in the action set.
- **Update metadata**: A metadata value can be associated with a packet. It is used to carry information from one table to the next. The Write-Metadata instruction updates an existing metadata value or creates a new value.

Flow Table Pipeline

A switch includes one or more flow tables. If there is more than one flow table, they are organized as a pipeline, with the tables labeled with increasing numbers starting with zero. The use of multiple tables in a pipeline, rather than a single flow table, provides the SDN controller with considerable flexibility.

The OpenFlow specification defines two stages of processing:

- **Ingress processing:** Ingress processing always happens, beginning with Table 0, and uses the identity of the input port. Table 0 may be the only table, in which case the ingress processing is simplified to the processing performed on that single table, and there is no egress processing.
- **Egress processing:** Egress processing is the processing that happens after the determination of the output port. It happens in the context of the output port. This stage is optional. If it occurs, it may involve one or more tables. The separation of the two stages is indicated by the numerical identifier of the first egress table. All tables with a number lower than the first egress table must be used as ingress tables, and no table with a number higher than or equal to the first egress table can be used as an ingress table.

Pipeline processing always starts with ingress processing at the first flow table; the packet must be first matched against flow entries of flow Table 0. Other ingress flow tables may be used depending on the outcome of the match in the first table. If the outcome of ingress processing is to forward the packet to an output port, the OpenFlow switch may perform egress processing in the context of that output port.

When a packet is presented to a table for matching, the input consists of the packet, the identity of the ingress port, the associated metadata value, and the associated action set. For Table 0, the metadata value is blank and the action set is null. At each table, processing proceeds as follows (see Figure 4.6):

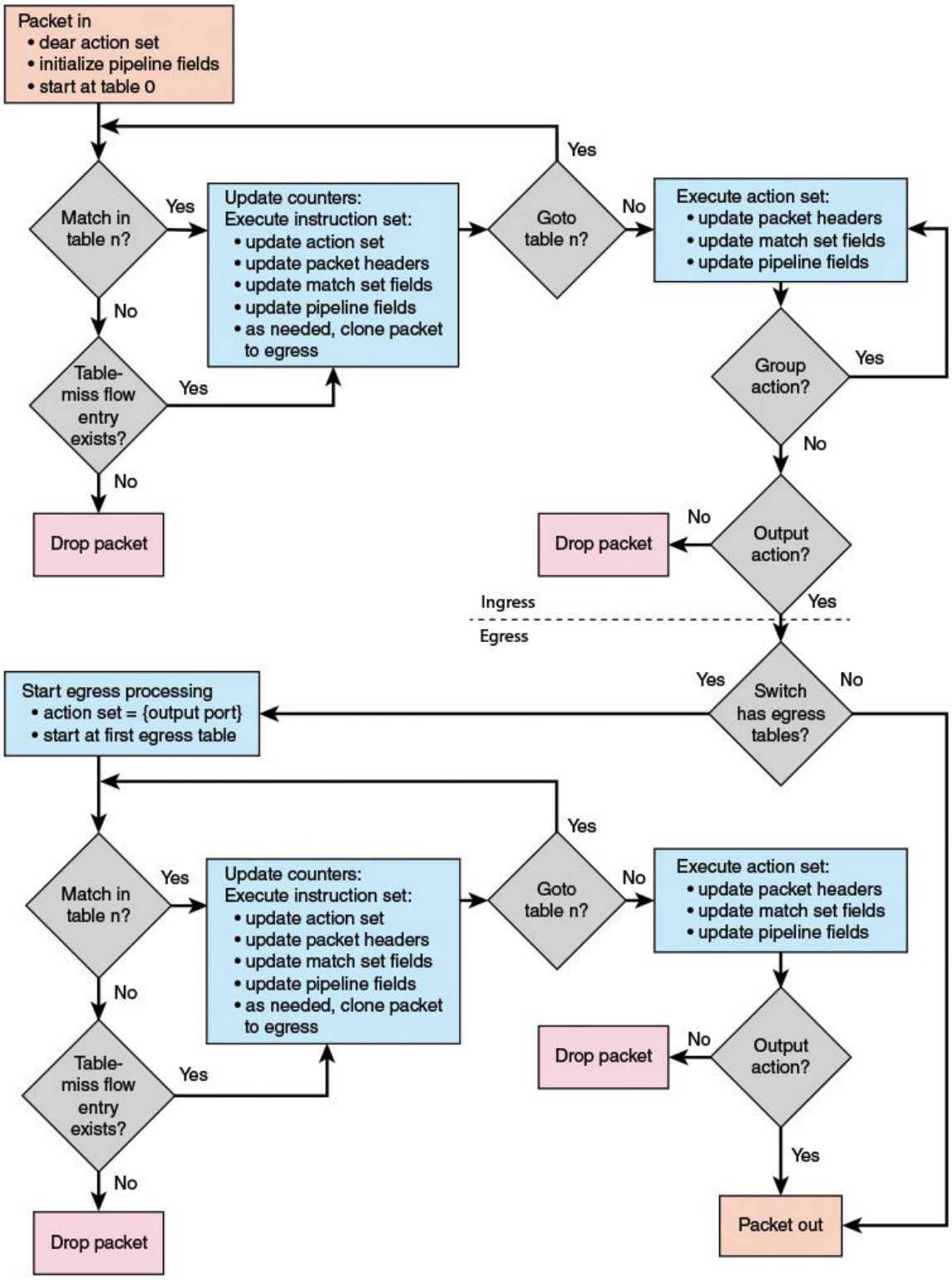

FIGURE 4.6 Simplified Flowchart Detailing Packet Flow Through an

1. If there is a match on one or more entries, other than the table-miss entry, the match is defined to be with the highest-priority matching entry. As mentioned in the preceding discussion, the priority is a component of a table entry and is set via OpenFlow; the priority is determined by the user or application invoking OpenFlow. The following steps may then be performed:

a. Update any counters associated with this entry.

b. Execute any instructions associated with this entry. This may include updating the action set, updating the metadata value, and performing actions.

c. The packet is then forwarded to a flow table further down the pipeline, to the group table, to the meter table, or directed to an output port.

2. If there is a match only on a table-miss entry, the table entry may contain instructions, as with any other entry. In practice, the table-miss entry specifies one of three actions:

a. Send packet to controller. This will enable the controller to define a new flow for this and similar packets, or decide to drop the packet.

b. Direct packet to another flow table farther down the pipeline.

c. Drop the packet.

3. If there is no match on any entry and there is no table-miss entry, the packet is dropped.

For the final table in the pipeline, forwarding to another flow table is not an option. If and when a packet is finally directed to an output port, the accumulated action set is executed and then the packet is queued for output. Figure 4.7 illustrates the overall ingress pipeline process.

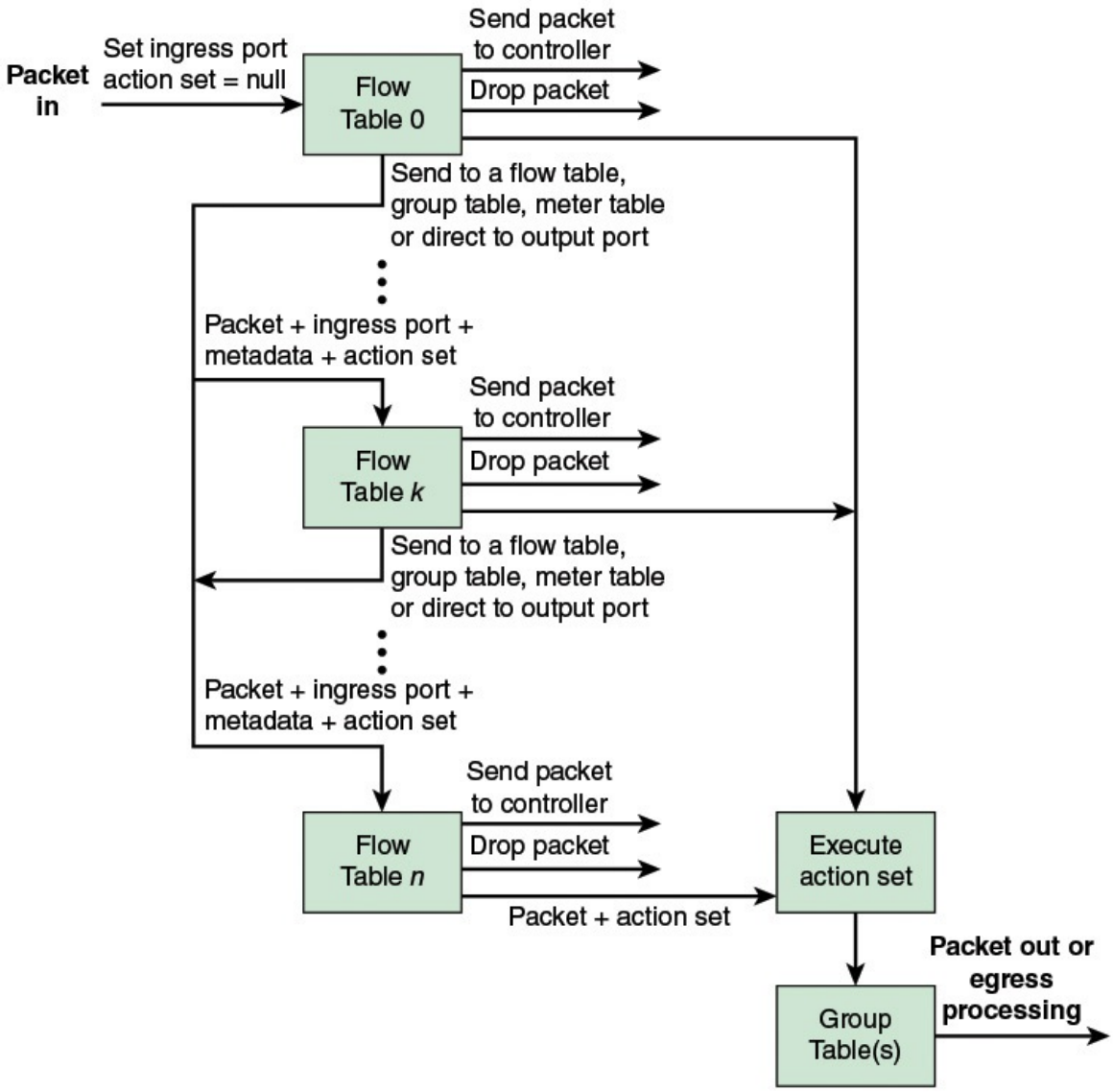

FIGURE 4.7 Packet Flow Through an OpenFlow Switch: Ingress Processing

If egress processing is associated with a particular output port, then after a packet is directed to an output port at the completion of the ingress processing, the packet is directed to the first flow table of the egress pipeline. Egress pipeline processing proceeds in the same fashion as for ingress processing, except that there is no group table processing at the end of the egress pipeline. Egress processing is shown in Figure 4.8.

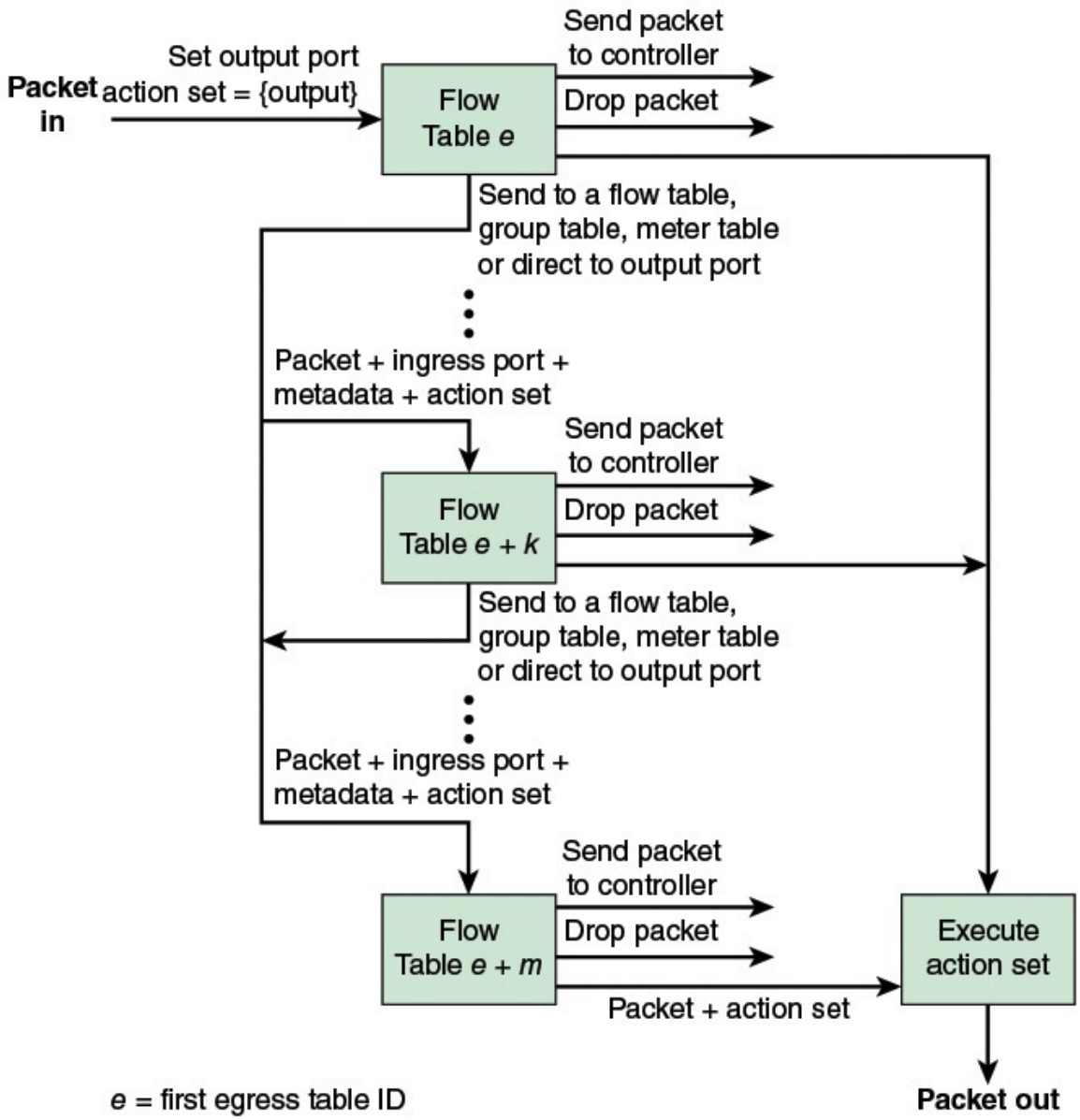

FIGURE 4.8 Packet Flow Through OpenFlow Switch: Egress Processing

The Use of Multiple Tables

The use of multiple tables enables the nesting of flows, or put another way, the breaking down of a single flow into a number of parallel subflows. Figure 4.9 illustrates this property. In this example, an entry in Table 0 defines a flow consisting of packets traversing the network from a specific source IP address to

a specific destination IP address. Once a least-cost route between these two endpoints is established, it might make sense for all traffic between these two endpoints to follow that route, and the next hop on that route from this switch can be entered in Table 0. In Table 1, separate entries for this flow can be defined for different transport layer protocols, such as TCP and UDP. For these subflows, the same output port might be retained so that the subflows all follow the same route. However, TCP includes elaborate congestion control mechanisms not normally found with UDP, so it might be reasonable to handle the TCP and UDP subflows differently in terms of quality of service (QoS)-related parameters. Any of the Table 1 entries could immediately route its respective subflow to the output port, but some or all of the entries may invoke Table 2, further dividing each subflow. The figure shows that the TCP subflow could be divided on the basis of the protocol running on top of TCP, such as Simple Mail Transfer Protocol (SMTP) or File Transfer Protocol (FTP). Similarly, the UDP flow could be subdivided based on protocols running on UDP, such as Simple Network Management Protocol (SNMP). The figure also indicates other subflows at Table 1 and 2, which may be used for other purposes.

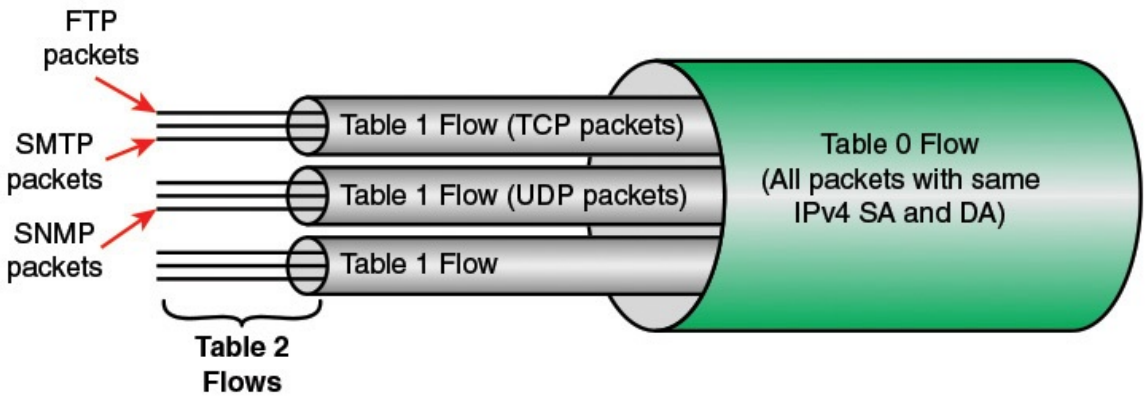

FIGURE 4.9 Example of Nested Flows

For this example, it would be possible to define each of these fine-grained subflows in Table 0. The use of multiple tables simplifies the processing in both the SDN controller and the OpenFlow switch. Actions such as next hop that apply to the aggregate flow can be defined once by the controller and examined and performed once by the switch. The addition of new subflows at any level involves less setup. Therefore, the use of pipelined, multiple tables increases the efficiency of network operations, provides granular control, and enables the network to respond to real-time changes at the application, user, and session levels.

Group Table

In the course of pipeline processing, a flow table may direct a flow of packets to the group table rather than another flow table. The group table and group actions enable OpenFlow to represent a set of ports as a single entity for forwarding packets. Different types of groups are provided to represent different forwarding abstractions, such as multicasting and broadcasting.

Each group table consists of a number of rows, called group entries, consisting of four components (refer back to part c of Figure 4.5):

- **Group identifier:** A 32-bit unsigned integer uniquely identifying the group. A **group** is defined as an entry in the group table.
- **Group type:** To determine group semantics, as explained subsequently.
- **Counters:** Updated when packets are processed by a group.
- **Action buckets:** An ordered list of action buckets, where each action bucket contains a set of actions to execute and associated parameters.

Each group includes a set of one or more action buckets. Each bucket contains a list of actions. Unlike the action set associated with a flow table entry, which is a list of actions that accumulate while the packet is processed by each flow table, the action list in a bucket is executed when a packet reaches a bucket. The action list is executed in sequence and generally ends with the Output action, which forwards the packet to a specified port. The action list may also end with the Group action, which sends the packet to another group. This enables the chaining of groups for more complex processing.

A group is designated as one of the types depicted in Figure 4.10: all, select, fast failover, and indirect.

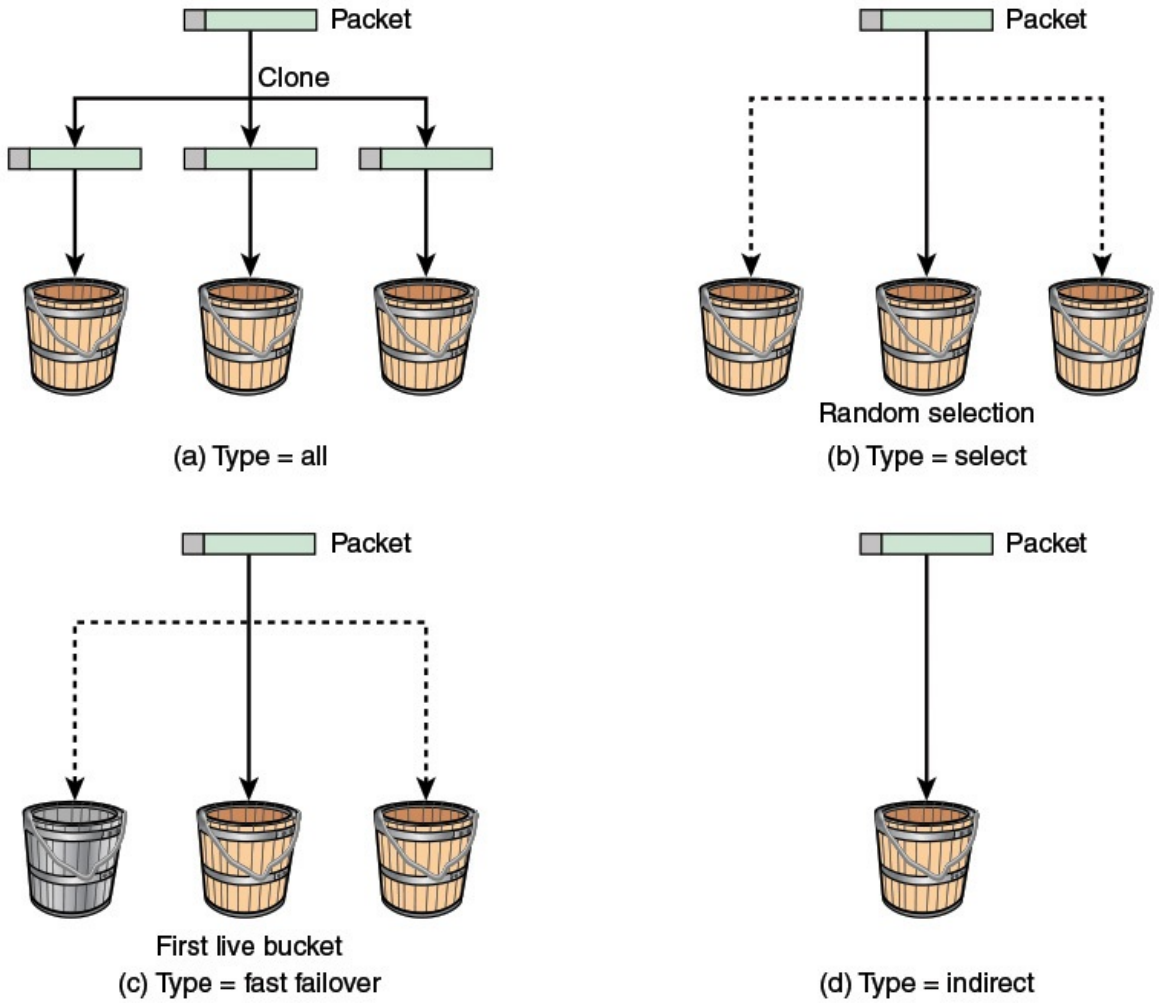

FIGURE 4.10 Group Types

The **all type** executes all the buckets in the group. Thus, each arriving packet is effectively cloned. Typically, each bucket will designate a different output port, so that the incoming packet is then transmitted on multiple output ports. This group is used for multicast or broadcast forwarding.

The **select type** executes one bucket in the group, based on a switch-computed selection algorithm (for example, hash on some user-configured tuple or simple round-robin). The selection algorithm should implement equal load sharing or, optionally, load sharing based on bucket weights assigned by the SDN controller.

The **fast failover type** executes the first live bucket. Port liveness is managed by code outside of the scope of OpenFlow and may have to do with routing algorithms or congestion control mechanisms. The buckets are evaluated in

order, and the first live bucket is selected. This group type enables the switch to change forwarding without requiring a round trip to the controller.

The three just-mentioned types all work with a single packet flow. The **indirect type** allows multiple packet flows (that is, multiple flow table entries) to point to a common group identifier. This type provides for more efficient management by the controller in certain situations. For example, suppose that there are 100 flow entries that have the same match value in the IPv4 destination address match field, but differ in some other match field, but all of them forward the packet to port X by including the action Output X on the action list. We can instead replace this action with the action Group GID, where GID is the ID of an indirect group entry that forwards the packet to port X. If the SDN controller needs to change from port X to port Y, it is not necessary to update all 100 flow table entries. All that is required is to update the group entry.

4.3 OpenFlow Protocol

The OpenFlow protocol describes message exchanges that take place between an OpenFlow controller and an OpenFlow switch. Typically, the protocol is implemented on top of TLS, providing a secure OpenFlow channel.

The OpenFlow protocol enables the controller to perform add, update, and delete actions to the flow entries in the flow tables. It supports three types of messages (see Table 4.2):

Message	Description
Controller to Switch	
Features	Request the capabilities of a switch. Switch responds with a features reply that specifies its capabilities.
Configuration	Set and query configuration parameters. Switch responds with parameter settings.
Modify-State	Add, delete, and modify flow/group entries and set switch port properties.
Read-State	Collect information from switch, such as current configuration, statistics, and capabilities.
Packet-out	Direct packet to a specified port on the switch.
Barrier	Barrier request/reply messages are used by the controller to ensure message dependencies have been met or to receive notifications for completed operations.
Role-Request	Set or query role of the OpenFlow channel. Useful when switch connects to multiple controllers.
Asynchronous-Configuration	Set filter on asynchronous messages or query that filter. Useful when switch connects to multiple controllers.
Asynchronous	
Packet-in	Transfer packet to controller.
Flow-Removed	Inform the controller about the removal of a flow entry from a flow table.

Port-Status	Inform the controller of a change on a port.
Role-Status	Inform controller of a change of its role for this switch from master controller to slave controller.
Controller-Status	Inform the controller when the status of an OpenFlow channel changes. This can assist failover processing if controllers lose the ability to communicate among themselves.
Flow-monitor	Inform the controller of a change in a flow table. Allows a controller to monitor in real time the changes to any subsets of the flow table done by other controllers.
Hello	Exchanged between the switch and controller upon connection startup.
Echo	Echo request/reply messages can be sent from either the switch or the controller, and must return an echo reply.
Error	Used by the switch or the controller to notify problems to the other side of the connection.
Experimenter	For additional functionality.

TABLE 4.2 OpenFlow Messages

- **Controller to switch:** These messages are initiated by the controller and, in some cases, require a response from the switch. This class of messages enables the controller to manage the logical state of the switch, including its configuration and details of flow and group table entries. Also included in this class is the Packet-out message. This message is sent by the controller to a switch when that switch sends a packet to the controller and the controller decides not to drop the packet but to direct it to a switch output port.
- **Asynchronous:** These types of messages are sent without solicitation from the controller. This class includes various status messages to the controller. Also included is the Packet-in message, which may be used by the switch to send a packet to the controller when there is no flow table match.
- **Symmetric**: These messages are sent without solicitation from either the controller or the switch. They are simple yet helpful. Hello messages are typically sent back and forth between the controller and switch when the connection is first established. Echo request and reply messages can be used by either the switch or controller to measure the latency or bandwidth of a controller-switch connection or just verify that the device is up and running. The Experimenter message is used to stage features to

be built in to future versions of OpenFlow.

In general terms, the OpenFlow protocol provides the SDN controller with three types of information to be used in managing the network:

- **Event-based messages:** Sent by the switch to the controller when a link or port change occurs.
- **Flow statistics:** Generated by the switch based on traffic flow. This information enables the controller to monitor traffic, reconfigure the network as needed, and adjust flow parameters to meet QoS requirements.
- **Encapsulated packets:** Sent by the switch to the controller either because there is an explicit action to send this packet in a flow table entry or because the switch needs information for establishing a new flow.

The OpenFlow protocol enables the controller to manage the logical structure of a switch, without regard to the details of how the switch implements the OpenFlow logical architecture.

4.4 Key Terms

After completing this chapter, you should be able to define the following terms.

action bucket

action list

action set

egress table

flow

flow table

group table

ingress table

match fields

OpenFlow action

OpenFlow instruction

OpenFlow message

OpenFlow port

OpenFlow switch

SDN data plane