5.6 OpenFlow 1.3 ADDITIONS

OpenFlow V.1.3 [8] was released on April 13, 2012. This release was a major milestone. Many implementations are being based on V.1.3 in order to stabilize controllers around a single version. This is also true about ASICs. Since V.1.3 represents a major new leap in functionality and has not been followed quickly by another release, this provides an opportunity for ASIC designers to develop hardware support for many of the V.1.3 features with the hope of a more stable market into which to sell their new chips. This ASIC opportunity notwithstanding, there are post V.1.0 features that are very challenging to implement in hardware. For example, the iterative matching actions we described in Section 5.4.1 are difficult to implement in hardware at line rates. It is likely that the real-life chips that support V.1.3 will have to limit the number of flow tables to a manageable number.[3] Such limitations are not imposed on software-only implementations of OpenFlow, and, indeed, such implementations will provide full V.1.3 support. Table 5.4 lists the major new features added in this release. A discussion of the most prominent of these new features is found in the following sections.

5.6.1 REFACTOR CAPABILITIES NEGOTIATION

There is a new MULTIPART_REQUEST/MULTIPART_REPLY message pair in V.1.3. This replaces the READ_STATE messaging in prior versions that used STATS_REQUEST/STATS_REPLY to get this information. Rather than having this capability information embedded as if it were a table statistic, the new message request-reply pair is utilized and the information is conveyed using a standard *type-length-value* (TLV) format. Some of the capabilities that can be reported in this new manner include *next-table*, *table-miss flow entry*, and *experimenter*. This new MULTIPART_REQUEST/REPLY message pair subsumes the older STATS_REQUEST/REPLY pair and is now used for both reporting of statistics as well as capability information. The data formats for the capabilities are the TLV format and this capability information has been removed from the table statistics structure.

[3] We discuss the possibility of ASICs that would allow more general support of these features in Section 14.9.3 and Section 15.3.7.

Table 5.4 Major New Features Added in OpenFlow 1.3

Feature Name	Description
Refactor Capabilities Negotiation	See Section 5.6.1
More Flexible Table Miss Support	See Section 5.6.2
IPv6 Extension Header Handling Support	See [8]
Per Flow Meters	See Section 5.6.3
Per Connection Event Filtering	See Section 5.6.4
Auxiliary Connections	See Section 5.6.5
MPLS BoS Matching	Bottom of Stack bit (BoS) from the MPLS header may now be used as part of match criteria
Provider Backbone Bridging Tagging	See Section 5.6.7
Rework Tag Order	Instruction execution order now determines tag order rather than it being statically specified
Tunnel-ID Metadata	Allows for support of multiple tunnel encapsulations
Cookies in PACKET_IN	See Section 5.6.6
Duration for Stats	The new *duration* field allows more accurate computation of packet and byte rate from the counters included in those statistics.
On Demand Flow Counters	Disable/enable packet and byte counters on a per-flow basis

5.6.2 MORE FLEXIBLE TABLE-MISS SUPPORT

We have explained that a table-miss is when a packet does not match any flow entries in the current flow table in the pipeline. Formerly there were three configurable options for handling such a table-miss. This included dropping the packet, forwarding it to the controller or continuing packet matching at the next flow table. V.1.3 expands upon this limited handling capability via the introduction of the *table-miss flow entry*. The controller programs a table-miss flow entry into a switch in much the same way it would program a normal flow entry. The table-miss flow entry is distinct in that it is by definition of lowest priority (zero) and all match fields are wildcards. The zero-priority characteristic guarantees that it is the last flow entry that can be matched in the table. The fact that all match fields are wild cards means that *any* packet being matched against the table-miss will be a match. The upshot of these characteristics is that the table-miss entry serves as a kind of backstop for any packets that would otherwise have found no matches. The advantage of this approach is that the full semantics of the V.1.3 flow entry, including instructions and actions, may be applied to the case of a table-miss. It should be obvious to the reader that the table-miss base cases of dropping the packet, forwarding it to the controller, or continuing at next flow table are easily implemented using the flow entry instructions and actions. Interestingly, though, by using the generic flow entry semantics, it is now conceivable to treat table-misses in more

sophisticated ways, including passing the packet that misses to a higher-numbered flow table for further processing. This capability adds more freedom to the matching "language" embodied by the flow tables, entries and associated instructions and actions.

5.6.3 PER FLOW METERS

V.1.3 introduces a flexible meter framework. Meters are defined on a per-flow basis and reside in a *meter table*. Fig. 5.15 shows the basic structure of a meter and how it is related to a specific flow. A given meter is identified by its *meter ID*. V.1.3 instructions may direct packets to a meter identified by its meter ID. The framework is designed to be extensible to support the definition of complex meters in future OpenFlow versions. Such future support may include color-aware meters that will provide DiffServ QoS capabilities. V.1.3 meters are only rate-limiting meters. As we see in Fig. 5.15 there may be multiple *meter bands* attached to a given meter. Meter 3 in the example in the figure has three meter bands. Each meter band has a configured bandwidth rate and a type. Note that the units of bandwidth are not specified by OpenFlow. The type determines the action to take when that meter band is processed. When a packet is processed by a meter, at most one band is used. This band is selected based on the highest bandwidth rate band that is lower than the current measured bandwidth for that flow. If the current measured rate is lower than all bands, no band is selected and no action is taken. If a band is selected, the action taken is that prescribed by the band's type field. There are no required types in V.1.3. The *optional* types described in V.1.3 consist of DROP and *DSCP remark*. DSCP remark indicates that the DiffServ drop precedence field of the *Differentiated Services Code Point* (DSCP)

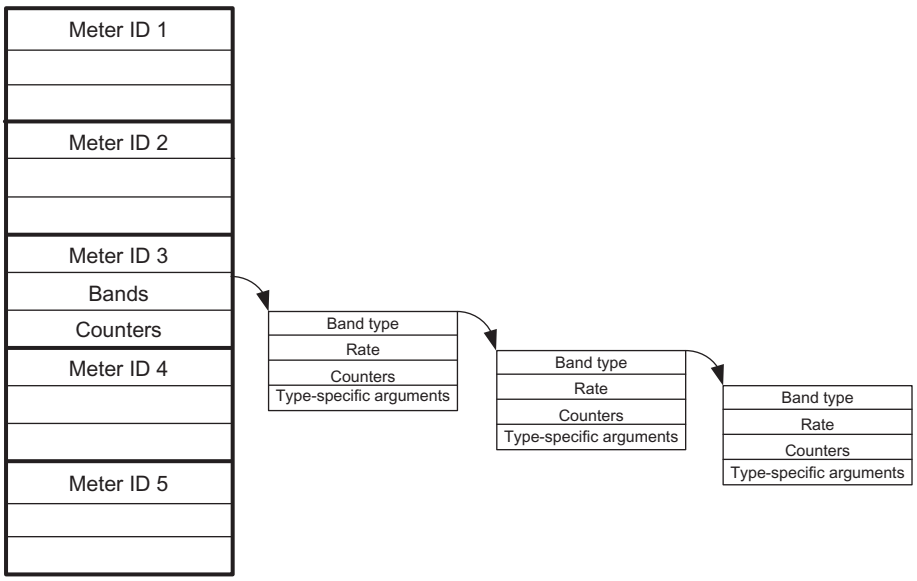


FIG. 5.15

OpenFlow V.1.3 meter table.

field should be decremented, increasing the likelihood that this packet will be dropped in the event of queue congestion. Well known and simple rate-limiting algorithms, such as the *leaky bucket* [9], may be implemented in a straightforward manner under this framework. This feature provides direct QoS control at the flow level to the OpenFlow controller by careful programming of the meters.

As seen in Fig. 5.15 there are meter-level counters and meter-band-level counters. The meter counters are updated for all packets processed by the meter and the per-meter-band counters are updated only when that particular band is selected. The dual level of counters is key in that presumably the majority of the packets processed by the meter suffer no enforcement, but their passage through the meter must be recorded in order to track the current measured rate. The reader should understand now that when a packet is processed by a band, it has exceeded a bandwidth threshold, which results in some kind of enforcement (e.g., dropping of the packet or marking it as *drop eligible* via DSCP remark).

5.6.4 PER CONNECTION EVENT FILTERING

We saw in V.1.2 the introduction of the notion of multiple controllers. Multiple controllers, able to take on different roles, achieve an improved level of fault tolerance and load balancing. The earlier discussion explained that the switches would only provide replies to the controller that initiated the request that elicits that reply. This, however, did not prescribe how to damp down the communication of asynchronous messages from the switch to its family of controllers. The idea that the different controllers have asymmetrical roles was not as effective as it could have been if all controllers must receive the same kind and quantity of asynchronous notifications from the switches. For example, a slave controller may not want to receive all types of asynchronous notifications from the switch. V.1.3 introduces a SET_ASYNC message that allows the controller to specify which sorts of async messages it is willing to receive from a switch. It additionally allows the controller to filter out certain reason codes that it does not wish to receive. A controller may use two different filters, one for the master/equal role and another for the slave role. Note that this filter capability exists in addition to the ability to enable or disable asynchronous messages on a per-flow basis. This new capability is controller-oriented rather than flow-oriented.

5.6.5 AUXILIARY CONNECTIONS

We have already seen that earlier releases of OpenFlow allowed some parallelism in the switch-controller channel via the use of multiple controllers. In this way, multiple parallel communication *channels* existed from a single switch to multiple controllers. V.1.3 introduces an additional layer of parallelism by allowing multiple *connections* per communications channel. That is, between a single controller and switch, multiple connections may exist. We depict these multiple connections between the switch and the MASTER controller in Fig. 5.16. The figure shows that there are also multiple channels connecting the switch to different controllers. The advantage provided by the additional connections on a channel lies in achieving greater overall throughput between the switch and the controller. Because of the flow-control characteristics of a TCP connection, it is possible that the connection be forced to quiesce (due to TCP window closing or packet loss) when there is actually bandwidth available on the physical path(s) between the switch and controller. Allowing multiple parallel connections allows the switch to take advantage of that. The first connection in the channel is specified to be a TCP connection. The specification indicates that other, nonreliable connection types,

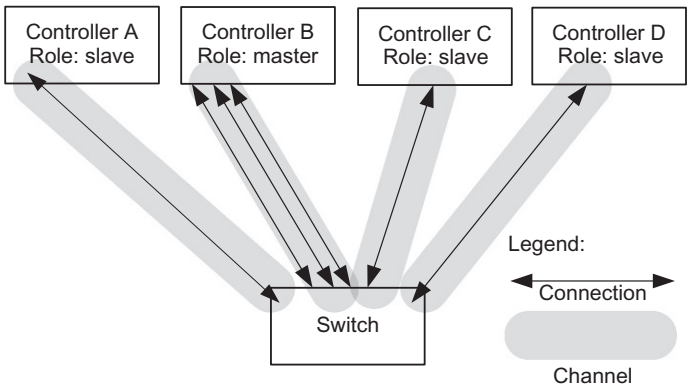


FIG. 5.16

Multiple connections and multiple channels from a switch.

such as UDP, may be used for the secondary connections, but the specification is fairly clear that the problems of potential data loss and sequencing errors that can arise from the use of such alternative transport layers is beyond the scope of the specification.

If bandwidth between the controller and switch becomes a constraint, it is most likely due to a preponderance of data packets. Some loss of data packets can be tolerated. Thus, the primary intended use of the auxiliary connections is to transmit and receive data packets between the switch and controller. This presumes that any control messages would be sent over the reliable, primary connection.

One example of the utility of auxiliary connections is that when a switch has many PACKET_IN messages to send to the controller, this can create a bottleneck due to congestion. The delay due to this bottleneck could prevent important OpenFlow control messages, such as a BARRIER_REPLY message, from reaching the controller. Sending PACKET_IN data messages on the UDP auxiliary connection will obviate this situation. Another possible extension to this idea is that the OpenFlow pipeline could send UDP-based PACKET_IN messages directly from the ASIC to the controller without burdening the control code on the switch CPU.

5.6.6 COOKIES IN PACKET-IN

The straightforward handling of a PACKET_IN message by the controller entails performing a complete packet matching function to determine what existing flow this packet relates to, if any. As the size of commercial deployments of OpenFlow grows, performance considerations have become increasingly important. Accordingly, the evolving specification includes some features that are merely designed to increase performance in high bandwidth situations. Multiple connections per channel, discussed above, is such an example. In the case of PACKET_IN messages, it is somewhat wasteful to require the controller to perform a complete packet match for every PACKET_IN message, considering that this look-up has already just occurred in the switch and, indeed, is the very reason the PACKET_IN message is being sent to the controller (that is, either a specific instruction to forward this packet to the controller was encountered or a table-miss resulted in the packet being handed off to the controller).

This is particularly true if this is likely to happen over and over for the same flow. In order to render this situation more efficient, V.1.3 allows the switch to pass a *cookie* with the PACKET_IN message. This cookie allows the switch to cache the flow entry pointed to by this cookie and circumvent the full packet matching logic. Such a cookie would not provide any efficiency gain the first time it is sent by the switch for this flow, but once the controller cached the cookie and pointer to the related flow entry, considerable performance boost can be achieved. The actual savings accrued would be measured by comparing the computational cost of full packet header matching versus performing a hash on the cookie. As of this writing, we are not aware of any study that has quantified the potential performance gain.

The switch maintains the cookie in a new field in the flow entry. Indeed, the flow entries in V.1.3 have expanded considerably as compared to the basic flow entry we presented in Fig. 5.5. The V.1.3.0 flow entry is depicted in Fig. 5.17. The header, counter and actions fields were present in Fig. 5.5 and were covered earlier in Section 5.3.2. The priority field determines where this particular flow entry is placed in the table. A higher priority places the entry lower in the table such that it will be matched before a lower priority entry. The timeouts field can be used to maintain two timeout clocks for this flow. Both an idle timer and a hard timeout may be set. If the controller wishes for the flow to be removed after a specified amount of time without matching any packets, then the idle timer is used. If the controller wishes for the flow to exist only for a certain amount of time regardless of the amount of traffic, the hard timeout is used and the switch will delete this flow automatically when the timeout expires. We provide an example of such a timeout condition in Section 5.3.6. We covered the application of the cookie field earlier in this section.

5.6.7 PROVIDER BACKBONE BRIDGING TAGGING

Earlier versions of OpenFlow supported both VLAN and MPLS tagging. Another WAN/LAN technology, known as *Provider Backbone Bridging* (PBB) is also based on tags similar to VLAN and MPLS. PBB allows for user LANs to be layer two bridged across provider domains, allowing complete domain separation between the user layer two domain and the provider domain via the use of *MAC-in-MAC* encapsulation. For the reader interested in learning more about PBB, we provide a reference

Header fields	Field value
Priority	Field value
Counters	Field value
Actions/instructions	Field value
Timeouts	Field value
Cookie	Field value

FIG. 5.17

V.1.3 flow entry.

to the related standard in [10]. This is a useful technology already supported by high-end commercial switches, and support for this in OpenFlow falls out easily as an extension to the support already provided for VLAN tags and MPLS shim headers.

5.6.8 EXAMPLE: ENFORCING QOS VIA METER BANDS

Fig. 5.18 depicts the new V.1.3 meter table as well as the group table that first appeared in V.1.1. We see in the figure that the incoming packet is matched against the second flow entry in the first flow table, and that the associated instructions direct the packet to Meter 3. The two dashed lines emanating from the meter table show that, based on the current measured bandwidth of the port selected as the output port (port *N* in this case), the packet may be dropped as exceeding the bandwidth limits, or it may be forwarded to port *N* if the meter does not indicate that bandwidth enforcement is necessary. As we have explained earlier, if the packet passes the meter control without being dropped or marked as dropped-eligible, it is *not* automatically forwarded to the output port, but may undergo any further processing indicated by the instructions in its packet processing pipeline. This could entail being matched against a higher-numbered flow table with its own instructions and related actions.

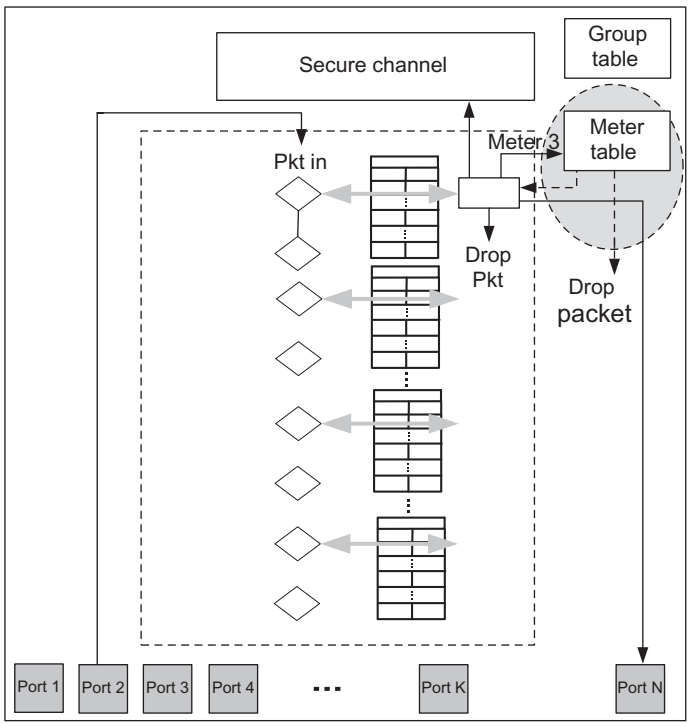


FIG. 5.18

Use of meters for QoS control in V.1.3.

DISCUSSION QUESTION:

Explain why, in Fig. 5.18, if the meter table does not drop the packet, the alternative dashed line indicates that the packet is not sent directly to the output port but rather forwarded back through the packet processing logic.

5.7 OpenFlow 1.4 ADDITIONS

OpenFlow V.1.4.0 [11] was released on October 14, 2013. Table 5.5 lists the major new features added in this release. In the case of the more significant features, the table description references one of the following sections where we provide more explanation of the feature. For other, more minor features, such as new error codes or clarification of the protocol, the table refers the interested reader to [11]. As of this writing, one switch manufacturer, Pica8, has already announced support for OpenFlow 1.4 [12]. The primary drivers behind this decision were the OpenFlow 1.4 support for bundles, eviction and vacancy events, and improved support for multiple controllers. Another notable feature of OpenFlow 1.4 was the new port descriptor which contained fields to support optical ports. We will provide further details on these more important features of OpenFlow 1.4 in the following sections.

5.7.1 BUNDLES

The bundle feature described in [11] provides enhanced transactional capabilities to the OpenFlow controller. In earlier versions, related, but more primitive control came from using a BARRIER_REQUEST

Table 5.5 New Features Added in OpenFlow 1.4

Feature Name	Description
More Extensible Wire Protocol	Standardize on TLV format: See [11]
More Descriptive Reasons for PACKET-IN	Clarification: See [11]
Optical Port Properties	See Section 5.7.4
Flow-removed reasons for meter delete	Clarification: See [11]
Flow monitoring	See Section 5.7.3
Role status events	See Section 5.7.3
Eviction	See Section 5.7.2
Vacancy Events	See Section 5.7.2
Bundles	See Section 5.7.1
Synchronized Tables	See Section 5.7.5
Group and Meter change notifications	See Section 5.7.3
Error code for bad priority	New Error Codes: See [11]
Error code for Set-async-config	New Error Codes: See [11]
PBB UCA header field	See [11]
Error code for duplicate instruction	New Error Codes: See [11]
Error code for multipart timeout	New Error Codes: See [11]
Change default TCP port to 6653	Official IANA port number assigned. See [11]

message whereby the switch must complete execution of all commands received prior to the BARRIER_REQUEST before executing any commands received after it. The OpenFlow 1.4 concept of a bundle is a set of OpenFlow messages that are to be executed as a single transaction. No component message should be acted upon by the switch until the entire set of commands in the bundle has been received and *committed*. The controller uses the new BUNDLE_CONTROL message to create, destroy or commit bundles. OpenFlow 1.4 also defines the two new commands BUNDLE_ADD_MESSAGE and BUNDLE_FAILED to add messages to a bundle and to report the failure of a bundle operation, respectively.

5.7.2 EVICTION AND VACANCY EVENTS

On physical switches, flow tables have very finite capacity and running out of room in a flow table on a switch can cause significant problems for the controller managing that switch. OpenFlow 1.4 contemplates a notion of *vacancy* whereby the switch can inform the controller that a critical level of table capacity has been reached. The motivation behind this is that by being given advance notice that the table is reaching capacity, the controller may be able to take logical steps to prune less important table entries and thus allow continued smooth operation of the switch. OpenFlow 1.4 allows the controller to act on this in a pro-active fashion by denoting certain flows as candidates for automatic eviction by the switch in the event that a vacancy threshold has been reached. Passing a vacancy threshold is communicated to the controller via a TABLE_STATUS message that includes reason codes supporting the vacancy feature. Parameters are provided to set the vacancy threshold itself as a means to quiesce flapping vacancy messages due to too-frequent passing of the threshold. Similarly, the eviction mechanism described in [11] allows configuration of various levels of importance to a flow entry to provide an orderly hierarchy of auto-eviction decisions taken by the switch.

DISCUSSION QUESTION:

Why are the eviction and vacancy events features more relevant to a ASIC-based switch than to a purely software-based switch?

5.7.3 ENHANCED SUPPORT FOR MULTIPLE CONTROLLERS

While OpenFlow 1.2 already included the concept of master and slave controllers, when one controller set itself to master, the previous master controller was not informed of this. In [11], a role status message is defined that allows the switch to inform a controller of a change to its master-slave role.

In addition, OpenFlow 1.4 permits definition of a set of monitors each of which may specify a subset of the flow tables on that switch. When a flow entry is added, deleted or changed within that subset, a notification is sent to the particular controller associated with that monitor. This capability is generically described as *flow monitoring* in [11].

Similarly, V.1.4 supports *group and meter change notifications* whereby the other controllers managing a switch are notified of group and meter changes effected by one controller.

5.7.4 OPTICAL PORT SUPPORT

The port descriptor is augmented in [11] to permit configuration and reporting on a number of optical-specific properties. These include fields for the transmit power of a laser, its transmit and receive frequencies or wavelengths, whether the transmitter and receiver are tunable, and whether the transmit power is configurable. These fields permit the configuration and monitoring of both Ethernet optical ports or the ports on optical circuit switches. Note that while these OpenFlow 1.4 extensions are a start for providing full OpenFlow support for optical ports, the ONF's *Optical Transport Working Group* (OTWG) has continued work in this area. The first specification emanating from this working group is discussed below in Section 5.10.

5.7.5 FLOW TABLE SYNCHRONIZATION

When two flow tables are synchronized, this signifies that when one table is changed the other table will be automatically updated. This is more complicated than it may first appear since the modification to one table may differ from that made to the second table. For example, if the first table matches on the source Ethernet address, the synchronization with the second table may transpose this to the destination Ethernet address. This is precisely the case for the example provided in [11] for flow table synchronization, which is a MAC learning bridge. In this example, when a new Ethernet address is learned on a port, an entry needs to be created in the first table (the learning table) with an idle timer so that inactive addresses can be later pruned. A *forwarding* table could be synchronized with the learning table and the newly learned (timed-out) addresses would be added (deleted) in that second table. The exact details of how the synchronization would occur are not specified in [11].

5.8 OpenFlow 1.5 ADDITIONS

OpenFlow V.1.5.0 [13] was released on December 19, 2014. Table 5.6 lists the major new features added in this release. In the case of the more significant features, the table description references one of the following sections where we provide more explanation of the feature. For other, more minor features related to consistency, flexibility or clarification of the protocol, the table refers the interested reader to [13]. A cursory inspection of the table of contents of the V.1.5 specification leads to the conclusion that there are far more additions to V.1.4 than is in fact the case. This is due to the fact that the document has been reorganized and section labels have been introduced to improved readability. There are, however several important new features introduced which we cover in the sections that follow.

5.8.1 ENHANCED L4–L7 SUPPORT

Even prior to OpenFlow 1.5, directing a packet to a logical port could be used to insert a network service into the flow. With the addition of *port recirculation* in V.1.5, it is now possible for the packet to be directed back to the OpenFlow packet processor after completing the logical port processing. This can enable *service chaining*, which is the sequencing of L4–L7 services such as firewalls or load balancers. This would occur by effectively chaining multiple logical ports such that a packet would be processed by each in sequence in the same switch.

Table 5.6 New Features Added in OpenFlow 1.5

Feature Name	Description
Egress Tables	See Section 5.8.3
Packet Type Aware Pipeline	See Section 5.8.4
Extensible Flow Entry Statistics	See Section 5.8.8
Flow Entry Statistics Trigger	See Section 5.8.8
Copy-Field action to copy between two OXM fields	See Section 5.8.2
Packet Register pipeline fields	See Section 5.8.2
TCP flags matching	See Section 5.8.1
Group Command for selective bucket operation	Flexibility: See [13]
Allow set-field action to set metadata field	See Section 5.8.1
Allow wildcard to be used in set-field action	See Section 5.8.1
Scheduled Bundles	See Section 5.8.5
Controller Connection Status	See Section 5.8.6
Meter Action	Flexibility: See [13]
Enable setting of all pipeline fields in PACKET_OUT	See Section 5.8.7
Port properties of all pipeline fields	See Section 5.8.7
Port Property for recirculation	See Section 5.8.1
Clarify and improve BARRIER	Consistency and clarification: See [13]
Always generate port status on port config change	See Section 5.8.6
Make all Experimenter OXM-IDs 64 bits	Consistency and clarification: See [13]
Unified requests for group, port and queue multiparts	Consistency and clarification: See [13]
Rename some types for consistency	Consistency and clarification: See [13]
Specification Reorganization	Consistency and clarification: See [13]

V.1.5 also offers the capability to maintain flow state via the storing and accessing flow metadata that persists for the lifetime of the flow, not just the current packet. Another new feature of V.1.5 that enhances L4 support is the ability to detect the start and end of TCP connections. This capability comes from the newly added capability of matching on the flag bits in the TCP header.

5.8.2 PIPELINE PROCESSING ENHANCEMENTS

The new *copy-field* action permits the copying of one header or pipeline field into another header or pipeline field. Prior to V.1.5, such fields could only be set to statically programmed values via the *set-field* action. *Packet registers*, new in V.1.5, provide scratchpad space that may be used to pass nonstandard information through the pipeline steps. These V.1.5 features, along with other, more minor ones appearing in Table 5.6, provide new robustness to the transformations that may occur to a packet as it passes through an OpenFlow pipeline.

5.8.3 EGRESS TABLES

V.1.5 introduces the notion of *egress* tables. In earlier versions, a flow table was consulted as part of the matching process on an incoming packet. This traditional role of the flow table is now referred to as an *ingress* table. Ingress table processing is still a mandatory part of the OpenFlow pipeline, whereas egress tables may or may not exist. In a switch where such two-stage pipeline processing is configured,

matching occurs against the egress tables once the output port(s) have been identified in the ingress processing. An example would be where the ingress processing consults a group table which results in the packet being forwarded out three output ports. By including egress tables, three distinct packet transformations may thus be performed in the separate contexts of each of the three output ports. This allows much greater flexibility in encapsulation than was formerly possible.

5.8.4 FITNESS FOR CARRIER USE

While a number of the V.1.5 features mentioned above improve OpenFlow's suitability in a carrier environment, most salient is the support of other data plane technologies beyond Ethernet. V.1.5 includes a new OXM pipeline field that identifies the packet type. The additional packet types supported in V.1.5 are IPV4 (no header in front), IPV6 (no header in front), no packet (e.g., circuit switch), and EXPERIMENTER.

5.8.5 BUNDLE ENHANCEMENTS

We introduced the concept of bundles in Section 5.7.1. V.1.5 now allows for an execution time to be specified with the bundle so that the bundle may be delivered to the switch in advance of the time that the switch should begin acting on the messages contained therein. This new version of OpenFlow also allows a controller to ask the switch to report its bundle capabilities, so that the controller will know what set of bundle features it may invoke on that switch.

5.8.6 ENHANCED SUPPORT FOR MULTIPLE CONTROLLERS

Some new features in V.1.5 are specifically targeted at facilitating more than one controller managing the same switch. One of these is *controller connection status*. This allows one controller to ask the switch the status of its connections to *any* controller, thereby affording one controller knowledge of other controllers managing the same switch. Another weakness in earlier versions was that when a port status was changed via OpenFlow, no port status change message was sent to the controller. In a single controller environment this is perfectly reasonable since the one controller would be aware that it itself effected the port status change. In a multicontroller environment, however, the remaining controllers would not be aware of the port status change which is potentially problematic. In V.1.5 a port status change message will always be sent to all controllers connected to the switch where the port was altered.

5.8.7 ENHANCED SUPPORT FOR TUNNELS

Starting with V.1.5, a logical port's properties allow specification of which OXM pipeline fields should be provided for a packet being sent to that port as well as which OXM fields should be included when a packet is transmitted from that logical port. This can provide enhanced tunnel support since the pipeline field *tunnel_id* can be used to hold the metadata associated for the encapsulation used by a tunnel. In support of this, V.1.5 also includes the ability to set *all* pipeline fields in the PACKET_OUT message, whereas earlier versions only supported the setting of the *in_port* field. In particular, should a logical port's V.1.5 properties denote that it *consumes* (i.e., requires) the *tunnel_id* field, the packet forwarding logic sending to that logical port would be able set the *tunnel_id* field thus passing information relevant to the aforementioned tunnel encapsulation.

5.8.8 ENHANCEMENTS TO FLOW ENTRY STATISTICS

Flow entry statistics are represented in a more extensible format starting in V.1.5. Statistic fields supported in earlier versions such as flow duration, and counters for flows, packets and bytes are now expressed in standard TLV format. Flow idle time is now captured as a standard statistic. In addition, support for arbitrary experimenter-based statistics is included. Another important performance enhancement pertaining to flow entry statistics is the addition of a trigger capability. This eliminates the overhead of the controller having to poll the switch for statistics and enables the switch to proactively send statistics to the controller on the basis of a trigger. This trigger may be based on either a time period or on the basis of a statistic reaching a certain threshold value.

5.8.9 SUMMARY

While OpenFlow V.1.3 provided the flexibility to handle most challenges posed by the data center, it did not meet the much greater flexibility required by network operators in the WAN [14]. The changes encompassed by V.1.4 and V.1.5 discussed above have provided many features needed for such WAN deployments. Commercial adoption of V.1.4 and V.1.5 has so far, however, been weak. Part of this stems from persistent problems achieving interoperability between mixes of controllers and switches from different manufacturers that all claim compliance with the standards. We discuss some of the ONF's approaches to address this issue in the next section.

5.9 IMPROVING OpenFlow INTEROPERABILITY

The move away from the single flow table of OpenFlow 1.0 to multiple flow tables, specified in OpenFlow 1.1, and implemented by most vendors in OpenFlow 1.3 was instigated in order to open up the rich potential of SDN to an ever increasing spectrum of applications. The single, very flexible flow table did not map well to the hardware pipelines available in advanced switching silicon [15]. While the chaining of multiple flow tables into a pipeline does indeed more closely approximate real hardware implementations, differences between these implementations required that any application wishing to use these detailed features needed to be developed with intimate knowledge of those hardware details. One reason for the difference in these implementations is that support for many OpenFlow 1.3 features is considered optional, further confusing the task for the application developer. The situation was such that applications intended for use on hardware switches needed to be developed specifically for those switches. While it was possible to implement uniform OpenFlow support on software-based switches, limiting an application to such switches usually reduces its usefulness.

In order resolve this interoperability problem, the ONF has espoused two new abstractions that are intended to facilitate interoperation between OpenFlow 1.3 switches. These two abstractions are *Table Type Patterns* (TTPs) and *Flow Objectives.* We discuss these in the following sections.

5.9.1 TABLE TYPE PATTERNS

In 2014, the ONF published the first TTP specification [16]. A TTP is a formal and detailed description of a complete set of logical switch behavior [15]. Many *OpenFlow Logical Switches* (OFLSs) may coexist inside a given physical switch. From an application developer's perspective, an OFLS possesses

the switching capabilities required by a particular application. From the switch vendor's perspective, the behavior of an OFLS is described succinctly in the OpenFlow specification terminology of the TTP. The fact that a particular TTP is succinct and limited to one application or class of applications makes it feasible for the switch vendor to confirm support or nonsupport for that particular OFLS. As the OFLS is described by a TTP, the TTP becomes the language that the controller and switch can use to confirm or deny the existence of a needed set of capabilities.

The formal language to describe a TTP is described in [15] and is based on *JavaScript Object Notation* (JSON). A very useful sample TTP describing an OFLS implementing a simple L2 and L3 forwarding switch is provided in [15].

5.9.2 FLOW OBJECTIVES

While TTPs provide a uniform way of describing a particular pipeline needed in a multi-table OpenFlow-enabled switch, the language of TTPs still requires detailed knowledge of the OpenFlow details that implement that pipeline. Flow objectives are intended to abstract multitable OpenFlow-level actions into generic application *objectives* [17]. These services-oriented objectives describe relatively high-level switching functions needed by an application but implemented via different pipelines on different physical switches. In Fig. 5.19 we show where flow objectives are defined within the *Open Network Operating System* (ONOS) embedded within the *Atrium* open source solution provided by the ONF [18]. Examples of these services-oriented objectives would include a filtering objective, a forwarding objective, and a next-hop objective.

The goal of exposing the flow objective abstraction is to permit application developers to build applications that benefit from scalable, multitable OpenFlow-based switching platforms without being aware of the details of the OpenFlow pipelines that would implement the needed services. As shown in Fig. 5.19, the pipeline-specific drivers architecturally below the flow objectives mask the differences between the different pipelines allowing the application developer to specify services at a more abstract level than using OpenFlow terminology.

DISCUSSION QUESTION:

Both TTPs and flow objectives are intended to enhance interoperability between OpenFlow implementations and as a result increase overall adoption of OpenFlow. In what ways do these two concepts differ in their approach to accomplishing these goals? In what way are they similar?

5.10 OPTICAL TRANSPORT PROTOCOL EXTENSIONS

The OpenFlow *Optical Transport Protocol Extensions* [19] was released on March 15, 2015. This specification builds upon the *port attribute extensibility* defined in OpenFlow 1.4. While that V.1.4 extension does facilitate the configuration and reporting of optical-specific port parameters, there are a number of areas that remain to be addressed to provide for robust OpenFlow support of practical optical networking. We discuss these below.

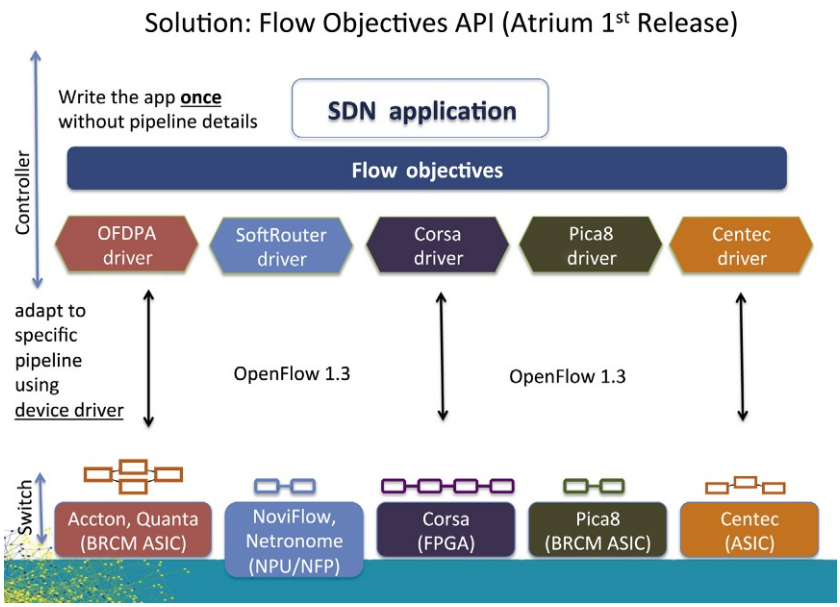


FIG. 5.19

Flow objectives abstraction.

Reproduced with permission from Saurav D. ATRIUM Open SDN Distribution, SDN Solutions Showcase, ONS 2015, Dusseldorf. Retrieved from: https://www.opennetworking.org/images/stories/news-and-events/sdn-solutions-showase/Atrium-ONS-Live.pdf.

5.10.1 MATCH/ACTION SUPPORT

In packet switching, much of the control functionality can be achieved by the control plane injecting control packets into the data plane. For example, knowledge about the adjacencies that together form the topology of a network can be collected by the exchange of such control packets between neighboring control planes. This method does not work with optical switching where the switches cannot simply inject packets in-line.

Similarly, optical switches are not packet switches, they are circuit switches. Implicit in this distinction, an OpenFlow-enabled circuit switch will not typically inspect packet headers to use as match criteria for flow tables, and thus not use that information to take actions such as forward or drop. Instead, the circuit switch will base these match and action decisions for a signal on the basis of an *Optical Channel* (OCh) characteristic such as wavelength for L0 switching or on the basis of an *Optical Data Unit* (ODU) characteristic like ODU tributary slot for L1 switching. The ODU tributary slot would identify the circuit by its position in time whereas individual physical optical channels are distinguished by transmission on different wavelengths. In either case, identification of the circuit permits a flow table-based set of rules to prescribe what action the switch should take for this circuit. Note that in this *Optical Transport Network* (OTN) context, we use the terms signal, channel, and circuit interchangeably.

5.10.2 OPTICAL PORT ATTRIBUTE SUPPORT

In [19] the OTWG defines *OTN port description extensions*. These extensions are used to identify and define the port types for L0 and L1 circuit switching control. More specifically, the extensions are used to specify and report on the characteristics of *Optical Transmission Section* (OTS), *Optical Physical Section* (OPS) and *Optical Channel Transport Unit* (OTU) ports. Examples of a characteristic that would be specified would be the specific layer class of an OTN signal. Examples of these layer classes are:

- OCH (OCh layer signal type)
- ODU (ODU layer signal type)
- ODUCLT (ODU Client layer signal type).

Within each of these layer classes, there are many possibilities for the specific signal type. This specific signal type is also specified in the OTN extensions. There are numerous other OTN-specific fields defined in these extensions and it is beyond the scope of this book to delve into the details of optical networking. We refer the reader to [19] for further details.

In [19] two methods are provided by which these OTN port description extensions may be passed between the controller and the switch, one based on OpenFlow 1.3 and one based on OpenFlow 1.4. One of the changes in OpenFlow 1.4 was extension of the basic port descriptor to provide support for a broader range of port types. While this expanded port descriptor provided a richer set of parameters for describing a port, it still lacks the full richness required to fully describe an OTN port. Thus, in OpenFlow 1.4 and beyond, the OTN extensions are captured by a combination of the extended port descriptor plus a set of OTN extensions defined in [19]. For OpenFlow 1.3, Ref. [19] specifies an experimenter multipart message whose body will contain the OpenFlow 1.4 port descriptor followed by those OTN extensions. (Note that these OTN extensions are sometimes referred to as the OTWG OpenFlow 1.5 OTN extensions.)

5.10.3 ADJACENCY DISCOVERY

Since optical switches cannot inject control packets in-line into the data plane to pass control and topology information through the network as packet switches currently do, other mechanisms are necessary to provide important basic features such as adjacency discovery. One proposed mechanism is based on using the control information known as the *Trail Trace Identifier* (TTI). Current ITU-sponsored standards provide that TTI be passed in-line within the data plane. In the optical world, this information is available to the control plane. The extensions contemplated in [19] allow for this TTI information to be made available via OpenFlow to the controller. The controller, in turn, can use this TTI information reported from different switches to synthesize adjacencies and thus network topology in much the way it does for packet switching. This represents a feasible mechanism for adjacency discovery for OpenFlow enabled optical circuit switches.

5.10.4 FUTURE WORK NEEDED

A number of areas that were identified as important for OpenFlow support of OTN require further specification than that provided in [19]. These include:

- Support for carrier reliability mechanisms for the data plane, in particular support for OAM monitoring of OTN network links as well as support for rapid protection switching functions that guarantee recovery from link failure.

- Support for network elements handling multiple technology layers according to transport layering models.
- Support for a hierarchy of controllers. In such a hierarchy, OpenFlow messages are exchanged between parent and child controllers in a *Control Virtual Network Interface* (CVNI) context. This is important as the scale required to support carrier-grade optical networks will require large numbers of OpenFlow controllers arranged in such a hierarchy.

As well as the future work identified in [19], the Calient corporation has identified areas that are urgently needed to foster the use of OpenFlow in optical circuit switches. These additional proposed extensions are described in [20].

DISCUSSION QUESTION:

Could OpenFlow-enabled switches support optical ports without the extensions described in Section 5.10? What major new class of OpenFlow-enabled switches is contemplated via these extensions?

5.11 OpenFlow LIMITATIONS

As OpenFlow remains in a state of rapid evolution, it is difficult to pin down precise limitations as these may be addressed in subsequent releases. One limitation is that the currently defined match fields are limited to the packet header. Thus, *Deep Packet Inspection*, (DPI) where fields in the packet's payload may be used to distinguish flows, is not supported in standard OpenFlow. Nonetheless, the EXPERIMENTER modes that are permitted within OpenFlow do open the way for such application-layer flow definition in the future. Secondly, some OpenFlow abstractions may be too complex to implement directly in today's silicon. This is unlikely to remain an insurmountable obstacle for long, however, as the tremendous momentum behind SDN is likely to give birth to switching chips that are designed explicitly to implement even OpenFlow's most complicated features. Another limitation is the possible processing delay if there is no matching entry in the OpenFlow switch's flow table, in which case processing the packet mandates that it be sent to the controller. While this is just a default behavior and the switch can be easily programmed to handle the packet explicitly, this potential delay is inherent to the OpenFlow paradigm. There are also a number of possible security vulnerabilities introduced by OpenFlow, such as (1) *Distributed Denial of Service* (DDoS) risks that can make the controller unavailable, and (2) failure to implement switch authentication in the controller. A brief discussion on these and other potential OpenFlow security vulnerabilities can be found in [21].

If we cast a wider net and consider limitations of all of Open SDN, there are a number of other areas to consider. We discuss these in Section 6.1.

5.12 CONCLUSION

In this chapter we have attempted to provide the reader with a high-level understanding of the general OpenFlow framework. This covered the protocol that an OpenFlow controller uses to configure and control an OpenFlow switch. We have also presented the switch-based OpenFlow abstractions that must be implemented on a physical switch as an interface to the actual hardware tables and forwarding engine of the switch in order for it to behave as an OpenFlow switch. For the reader interested in a

Table 5.7 OpenFlow Protocol Constant Classes

Prefix	Description
OFPT	OpenFlow Message Type
OFPPC	Port Configuration Flags
OFPPS	Port State
OFPP	Port Numbers
OFPPF	Port Features
OFPQT	Queue Properties
OFPMT	Match Type
OFPXMC	OXM Class Identifiers
OFPXMT	OXM Flow Match Field Types for Basic Class
OFPIT	Instruction Types
OFPAT	Action Types
OFPC	Datapath Capabilities and Configuration
OFPTT	Flow Table Numbering
OFPFC	Flow Modification Command Type
OFPFF	Flow Modification Flags
OFPGC	Group Modification Command Type
OFPGT	Group Type Identifier
OFPM	Meter Numbering
OFPMC	Meter Commands
OFPMF	Meter Configuration Flags
OFPMBT	Meter Band Type
OFPMP	Multipart Message Types
OFPTFPT	Flow Table Feature Property Type
OFPGFC	Group Capabilities Flags

deeper understanding of these issues, whether out of sheer academic interest or a need to implement one of the versions of OpenFlow, there is no substitute for reading the OpenFlow specifications cited in this chapter, especially [13], as it is generally an updated superset of its predecessors. These specifications are, however, very detailed and are not especially easy for the OpenFlow novice to follow. We believe that this chapter will serve as an excellent starting point and guide for that reader who needs to delve into the lengthy specifications that comprise the OpenFlow releases covered here. In particular, these specifications contain a plethora of structure and constant names. To aid in reading the specifications, we draw your attention to Table 5.7 in this chapter as a quick reference and aid to understanding to what part of the protocol a specific reference pertains.

REFERENCES

[1] OpenFlow Switch Specification, Version 1.0.0 (Wire Protocol 0x01). Open Networking Foundation, December 31, 2009. Retrieved from: https://www.opennetworking.org/sdn-resources/onf-specifications.

[2] IEEE Standard for Local and Metropolitan Area Networks—Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks—Amendment 21: Edge Virtual Bridging. IEEE 802.1Qbg, July 2012. New York: IEEE; 2012.

[3] OpenFlow Switch Specification, Version 1.1.0 (Wire Protocol 0x02). Open Networking Foundation, December 31, 2009. Retrieved from: https://www.opennetworking.org/sdn-resources/onf-specifications.
[4] Davie B, Doolan P, Rekhter Y. Switching in IP networks. San Francisco: Morgan Kaufmann; 1998.
[5] IEEE Standard for Local and Metropolitan Area Networks–Link Aggregation. IEEE 802.1AX, November 3, 2008. USA: IEEE Computer Society; 2008.
[6] OpenFlow Switch Specification, Version 1.2.0 (Wire Protocol 0x03). Open Networking Foundation, December 5, 2011. Retrieved from: https://www.opennetworking.org/sdn-resources/onf-specifications.
[7] IEEE Standard for Local and Metropolitan Area Networks—Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks. IEEE 802.1Q, August 2011. New York: IEEE; 2011.
[8] OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04). Open Networking Foundation, June 25, 2012. Retrieved from: https://www.opennetworking.org/sdn-resources/onf-specifications.
[9] Shenker S, Partridge C, Guerin R. Specification of guaranteed quality of service. RFC 2212. Internet Engineering Task Force; 1997.
[10] Draft Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks—Amendment 6: Provider Backbone Bridges. IEEE P802.1ah/D4.2, March 2008. New York: IEEE; 2008.
[11] OpenFlow Switch Specification, Version 1.4.0 (Wire Protocol 0x05). Open Networking Foundation, October 14, 2013. Retrieved from: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.4.0.noipr.pdf.
[12] Oliver B. Pica8: First to adopt openflow 1.4; why isn't anyone else? Tom's ITPRO, May 2, 2014. Retrieved from: http://www.tomsitpro.com/articles/pica8-openflow-1.4-sdn-switches,1-1927.html.
[13] OpenFlow Switch Specification, Version 1.5.0 (Wire Protocol 0x06). Open Networking Foundation, December 19, 2014. Retrieved from: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.
[14] Hunt G. The Quest for Dominance: OpenFlow or NETCONF for Networks Outside the Data Center? Network Matter, February 6, 2015. Retrieved from: http://networkmatter.com/2015/02/06/the-quest-for-dominance-openflow-or-netconf-for-networks-outside-the-data-center/.
[15] Dixon C. OpenFlow & Table Type Patterns in OpenDaylight's Next Release. SDX Central, August 11, 2014. Retrieved from: https://www.sdxcentral.com/articles/contributed/openflow-table-type-patterns-opendaylight-next-release-colin-dixon/2014/08/.
[16] OpenFlow Table Type Patterns, Version 1.0 (ONF TS-017). Open Networking Foundation, August 15, 2014. Retrieved from: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/OpenFlowTableTypePatternsv1.0.pdf.
[17] Pitt D. Paving the way For SDN interoperability. InformationWeek Network Computing, October 9, 2015. Retrieved from: http://www.networkcomputing.com/networking/paving-the-way-for-sdn-interoperability-/a/d-id/1322562.
[18] Saurav D. ATRIUM Open SDN Distribution. SDN Solutions Showcase, ONS 2015, Dusseldorf. Retrieved from: https://www.opennetworking.org/images/stories/news-and-events/sdn-solutions-showcase/Atrium-ONS-Live.pdf.
[19] Optical Transport Protocol Extensions, Version 1.0 (ONF TS-022). Open Networking Foundation, March 15, 2015. Retrieved from: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/Optical_Transport_Protocol_Extensions_V1.0.pdf.
[20] Optical Circuit Switch OpenFlow Protocol Extensions. Calient, June 2, 2015. Retrieved from: https://wiki.opendaylight.org/images/1/1d/OCS_OF_Protocol_Extensions_Rev._0.4.pdf.
[21] Benton K, Camp L, Small C. OpenFlow Vulnerability Assessment. HotSDN, SIGCOMM 2013, August 2013, Hong Kong. Retrieved from: http://conferences.sigcomm.org/sigcomm/2013/papers/hotsdn/p151.pdf.