The Evolution of SDN and OpenFlow: A Standards Perspective

Jean Tourrilhes; Puneet Sharma; Sujata Banerjee; Justin Pettit

HP Laboratories
HPL-2014-41

Keyword(s):

SDN; OpenFlow; Networking

Abstract:

SDN is designed to address networking needs that are poorly addressed by existing networks, and therefore the OpenFlow protocol and its specification process are significantly different from most traditional network protocols. The evolution of the OpenFlow API is guided by a large number of unrelated use cases, the main elements of SDN and classical guidelines of API development. However the goal of addressing a wide range of network devices, hardware and software, and the volunteer process in the Extensibility WG make the specification process quite complex. To address these challenges, the specification process for OpenFlow was made very dynamic and quite similar to the process of open source projects, and the deliverables include both a stable long term support branch and a more experimental main branch.

External Posting Date: December 6, 2014 [Fulltext] Approved for External Publication
Internal Posting Date: December 6, 2014 [Fulltext]

The Evolution of SDN and OpenFlow: A Standards Perspective

Jean Tourrilhes, Puneet Sharma, Sujata Banerjee – HP-Labs - {FirstName.LastName}@hp.com

Justin Pettit – VMware – jpettit@vmware.com

1. Introduction

Software Defined Networking (SDN) is arguably one of the most significant paradigm shifts the networking industry has seen in recent years. SDN has been driven by the need to keep pace with new network requirements from emerging trends of virtualized cloud computing, mobility and big data applications. The goals of SDN include the ability to introduce network innovations faster and to radically simplify and automate the management of large networks [1]. However, most principles behind SDN are not entirely new - for example network programmability was experimented with Active Networking in the 1990s and separation of control plane was introduced by IETF ForCES Working Group (WG) [2] in the 2000s. Unfortunately, a lot of that research was never widely deployed, for a variety of good reasons [3].

The technical and business history of SDN and OpenFlow (the predominant SDN protocol) has been described well in [3] and other prior papers. It is not our intention to rehash the same history in this paper. Instead, the purpose of this paper is to present our perspective on how elements of the SDN framework and the OpenFlow protocol features have evolved since their inception, and demystify the standardization process around OpenFlow in the Open Networking Foundation (ONF). Within the ONF, the Extensibility WG is responsible for maintaining and evolving the OpenFlow protocol [3]. The work carried out in the Extensibility WG is of prime importance in realizing the SDN promise of faster introduction of network innovations. In our view, the evolution of SDN has been fundamentally driven by the most pressing or promising use cases.

2. The OpenFlow specification process

Prior to the creation of the ONF in 2011, the OpenFlow specification was managed by a loose group of individuals meeting physically at Stanford University [1]. The first task of the Extensibility WG was to define a new process enabling ONF member companies to participate in the evolution of OpenFlow without losing the initial momentum. A more formal process was needed for better transparency and accountability, and to better collaborate with other workgroups in the ONF; however it was essential to retain the flexibility and dynamism of the earlier development.

The process in the Extensibility WG for each proposal contains four phases: initial proposal presentation, incubation, prototyping and a final ONF wide review. During the proposal presentation phase, the use case is presented alongside a set of technical solutions. The incubation phase is when the actual specification text for the proposal is written, reviewed and approved. An open source prototype of the proposal is implemented, reviewed and approved during the prototyping phase. All approvals are done by rough consensus during one of the weekly conference calls. Rough consensus is achieved after every opinion has been expressed and discussed, and there is an absence of major objections.

The last phase, the ONF-wide review, allows other workgroups, the Technical Advisory Group (TAG), the Chip Advisory Board (CAB) and member companies to review the full specification before final approval in the Extensibility WG. After that, the final specification must be approved by the Board of ONF before external publication.

The ONF has adopted this process with minor variations in other workgroups, such as the Configuration and Management WG [15] and the Forwarding Abstraction WG [14].

A goal of ONF is to foster implementations in the marketplace. The strong open source ecosystem built around version 1.0 of OpenFlow is one of the reasons for its success and longevity. Multiple open source implementations of switches and controllers were available, as well as many useful tools. When forming the Extensibility WG, one of the goals was to reduce the gap between the open source community outside ONF, the vendors and the specification process. The Extensibility WG invited various developers into the specification process and specified many experimental features of open source or vendor implementations in newer versions of OpenFlow. The specification process was also amended to mandate open source prototyping of new features, so that the community could have experience with the implementation of the new features before standardization was completed. The ecosystem around version 1.3 and later has now surpassed version 1.0 and is growing.

3. Example SDN Use Cases

SDN use cases are the driving force behind the evolution of OpenFlow. The number of use cases based on OpenFlow has grown over the years, and it is not possible to list them all. In this section, we selected two use cases that were amongst the most influential on the recent evolution of OpenFlow and that are the most complete and best illustrate the diverse scope of issues the SDN framework had to address, and the OpenFlow feature set they influenced. The two use cases are: (1) Cloud datacenters and (2) Unified Communications in Enterprises. We describe their effect on the SDN framework elements and the OpenFlow features they required in the following section.

3.1. Cloud datacenter

Computing in the past decade has gone through a significant transformation. In the same way that servers replaced mainframes, dedicated servers are being replaced by virtualized cloud datacenters, offering increased flexibility and lower costs. This transformation is built on commodity hardware and software; the enablers are computer virtualization and automation middleware [4].

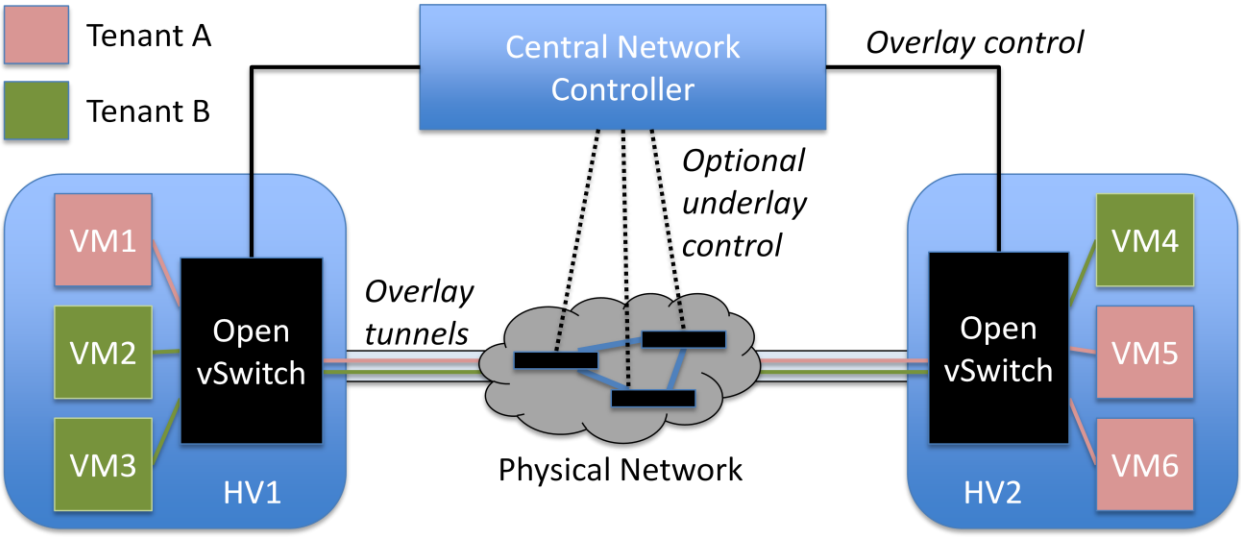

Figure 1: Network virtualization provides logical tenant networks over a shared physical network. The central network controller manages the vSwitches to implement this application using a protocol such as OpenFlow.

Computing resources in cloud datacenters are instantiated automatically in a matter of minutes. In contrast, typical network management is manually performed by a human often using the command-line-interface (CLI) of each network element, and thus much slower. Network outages can have wide impact, and the impact of network changes is complex to predict. Since diagnosing network behaviour is difficult, most network changes are done carefully by a human.

One approach to this new dynamic paradigm is to keep the network decoupled from computing volatility, and provide only a flat static communication service. However, Cloud datacenters put new requirements on the underlying network. For instance, the traffic from different tenants needs to be segregated for both security and performance. Various network functions, such as firewalling, deep packet inspection (DPI) and load balancing need to be inserted on-demand and in-line with the tenant traffic. Thus networking functions need to be even more coupled to computing functions than ever, as the network policies must match the compute policies, and the usual static configuration of network cannot suffice.

A common solution today is to deploy an SDN overlay to map this dynamic configuration to the static network [5]. In Figure 1, a multi-tenant datacenter environment is depicted with the network virtualization solution components. In each server, the vSwitch, a software network switch, dynamically routes the packets from the virtual machines (VMs) on different static tunnels established over the network (see Figure 1). The datacenter manager has an API to communicate the new connectivity requirements to the network controller when making changes to the compute placement. The network controller can then use an API, such as OpenFlow, to implement the connectivity requirements and policies in the vSwitch.

To some people, the current SDN overlay solutions, decoupled from the physical underlay, are only a partial solution [6]. Enabling the network controller to tightly manage the hardware switches in

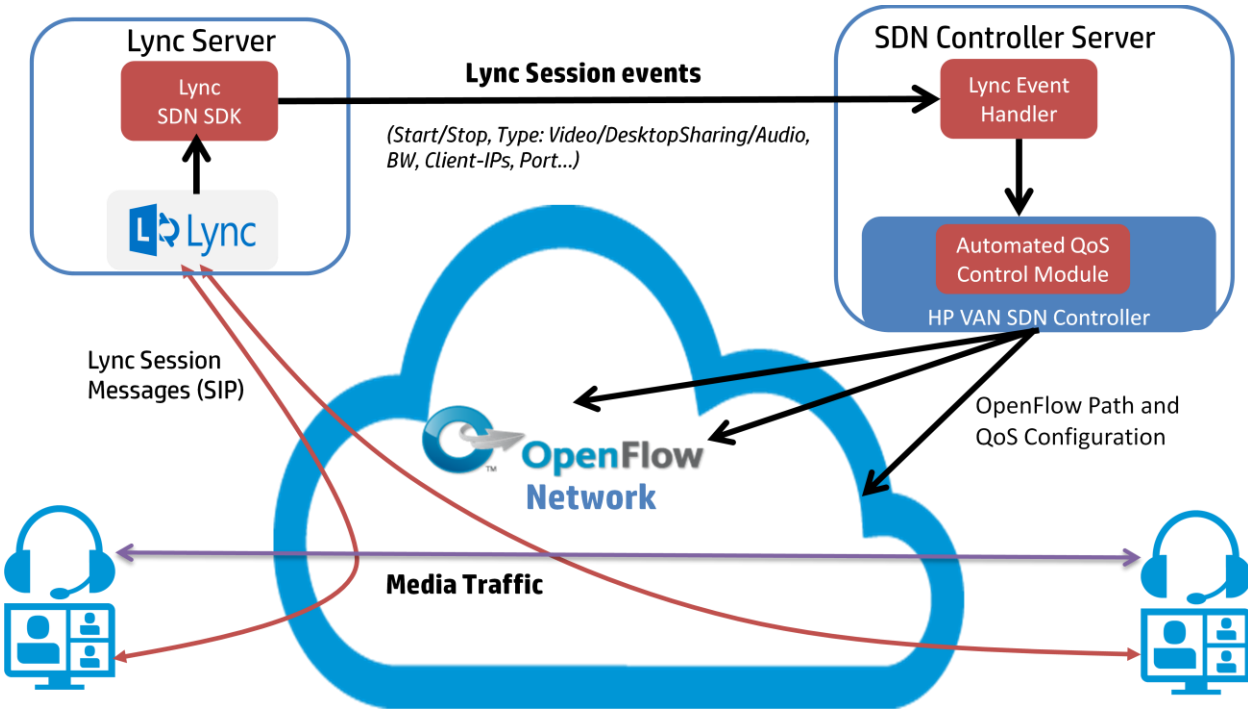

Figure 2: Unified communication with Lync and OpenFlow

addition to the vSwitch would avoid the need for the controller to infer the network behaviour and instead to explicitly provision its needs. For example, VMware and HP have demonstrated a solution where the controller provisions and jointly manages elephant flows in the overlay and the underlay networks [6]. Another approach for a more predictable and scalable datacenter network is to eliminate all learning and discovery from the network [7]. In this scenario core switch functions, such as forwarding rules and tunnel configuration, are programmed via OpenFlow using the datacenter central database.

Multipath support added in version 1.1 is an example of OpenFlow evolution driven by the desire to leverage OpenFlow in Cloud Datacenters [8]. Encapsulation is a basic primitive to create overlay tunnels, and work in the Extensibility WG led to the inclusion of overlay encapsulation metadata in version 1.3 and generic encapsulation is being worked on for version 1.5.

3.2. Unified Communications in Enterprises

Interactive media applications like video conferencing or remote desktop that require Network Quality of Service (QoS) to guarantee application performance are either deployed on costly dedicated networks or suffer from poor quality on non-dedicated purely best-effort networks. These applications fundamentally require QoS to be maintained along the whole end-to-end network path based on global network wide policy, making it complex to deploy and configure. Hence, existing network QoS implementations are primarily limited to QoS marking on the edge based on a priori static decisions with partial network view. Today's networks do not have efficient ways to identify flows requiring QoS, and

which policy applies to each of them. Additionally, an admission control system needs compute and configure appropriate QoS upon QoS flow arrival. The QoS provisioning should also be dynamically adapted to meet varying traffic demands and QoS sensitive workloads.

SDN can help to automate network-wide QoS configurations for Unified Communications sessions, as demonstrated by the OpenFlow Lync demonstration created by HP and Microsoft [9]. Figure 2 depicts how an SDN and OpenFlow enabled network can automatically provision QoS flows. Microsoft Lync is an enterprise communications and messaging product including audio-video conferencing, desktop-sharing and shared-whiteboard using end-client on PC and central Lync server to manage communication sessions and policies. The HP Virtual Application Network (VAN) controller is an SDN controller that can configure network switches using the OpenFlow protocol. The Lync server and the HP VAN controller interact with each other through a new SDN northbound API. A QoS module on the SDN controller, monitors network resource usage and can apply specific QoS treatment to network flows based on QoS policies [10]. The QoS module has a global view of both the network and the demand, so it can easily map global QoS policies to the network.

When a new Lync session is initiated, the Lync server can communicate the requirements of the session such as required bandwidth and end-to-end latency to check with the SDN controller if resources are available (see Figure 2). The QoS controller module can then determine the policy applicable to that connection, based on global QoS policies, the attributes given by the Lync server, the state of other QoS connections placed on the network and optionally based on user identity (retrieved using a directory service such as LDAP). Then, the QoS module programs this policy on the various switches along the path using OpenFlow. The goal is to not only provision required QoS but also maximize network resource usage. If session's perceived quality is not acceptable, the network state can be analyzed in real time and specific actions (example, rerouting the call) can be taken. Enterprise networks with autonomous control are amenable to such global QoS control.

The development of the QoS module necessitated the creation of QoS extensions to OpenFlow [10], and these new APIs were eventually added in a more generic form into OpenFlow. Matching on priority code point (PCP) and differentiated services code point (DSCP) fields was added in version 0.8.9, and queue selection was added in version 1.0. Similarly, a feature to support meters (rate limiters and policers) required by the QoS controller module to police the traffic at the network edge was added in version 1.3 after being initially introduced as a vendor extension.

4. SDN elements

The two use cases presented in the previous section are quite different, but both leverage OpenFlow to meet their requirements. It would be possible to develop a point solution for each of them, addressing narrowly only that particular use case. Such an approach would lead to a proliferation of specialized protocols, one for each use case, and a proliferation of interfaces to configure those protocols. If we want to address the large set of use cases and policies with minimal duplication of efforts, a generic framework must be defined based on the common elements to those use cases. We

consider the following five elements of the SDN architectural framework listed in Table 1, and describe each element in turn.

SDN Element	OpenFlow switch specification improvements	OF version
Separation of control plane	Cookies - attach policy identifier to flow entries	1.0
	Vacancy events - control table resource policy	1.4
	Delegation of flow learning to the switch	1.5 (proposed)
Logical centralization	Role election mechanism (multi-controller)	1.2
	Event filtering (multi-controller)	1.3
	Bundles – apply a set of requests together (fate sharing)	1.4
	Flow monitoring (multi-controller)	1.4
Northbound APIs	Per port priority queues – per application QoS	1.0
	Per flow meters – per application QoS	1.3
	Tunnel-id field – support for overlay on logical ports	1.3
Programmability	Groups – they are reusing flow actions	1.1
	Extensible fields common to matching and rewriting headers	1.2
	Table features (capability expression)	1.3
Flow entries	Multiple tables of flow entries	1.1
	Egress tables – output processing using flow entries	1.5 (proposed)
	Encapsulation action – tunnelling using flow entries	1.5 (proposed)

Table 1: Five SDN elements and some example OpenFlow specification changes related to them

4.1. Separation of control plane

Separation of control plane means that the decision about how to handle traffic is not made by the entity handling the traffic, and all policy decisions in the network are made by a centralized controller [1]. The goal is to keep the policies separate from the network device and management APIs, this avoids the need to standardize policies and enables deployments to be free to invent and implement any policy of their choosing. The controller must have full and absolute control over the network device, so that the desired policies can be properly implemented without ambiguity.

Separation of control requires a well-defined and standard API between the controller and the network device, so that the two can be logically separated. OpenFlow is defined as a network protocol so that the physical separation can also be achieved on top of the logical separation. The controller owns all the network policies, and uses the mechanisms in the API to enforce those policies. The OpenFlow protocol must therefore offer enough control and visibility to enforce those policies.

The requirement to avoid standardizing policies forces OpenFlow to expose the functionality of the switch at a much lower level than most existing management interfaces, and it exposes directly packet processing. The switches expose an OpenFlow pipeline that maps to its hardware packet processing and can dictate where packets are sent and how packets are modified. For packets processed by OpenFlow, most traditional control functions are bypassed, unless explicitly requested.

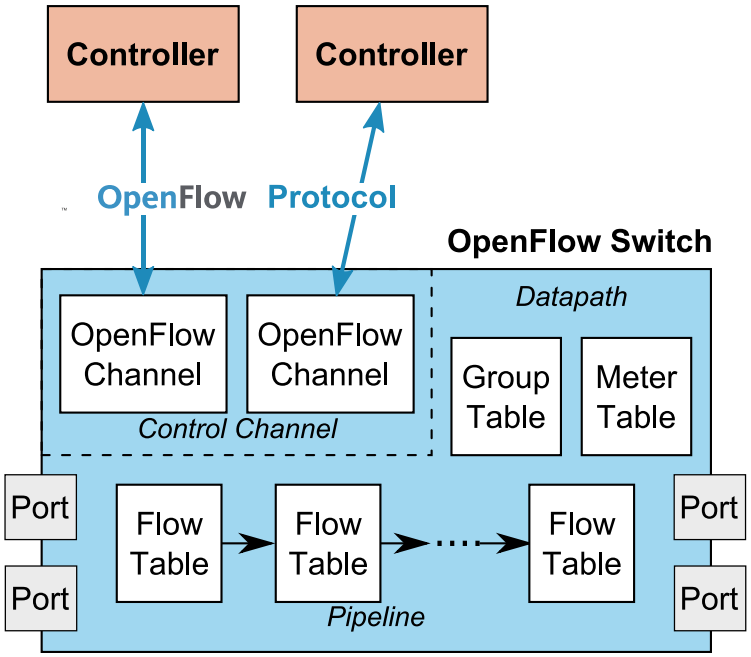

Figure 3: An OpenFlow switch and a logically centralised controller

The split of responsibility between the controller and the network device is evolving. Complex processing may be offloaded to the datapath [11] – however, the policy decision must rest in the control function. For example, delegation of learning to the switch has been proposed for version 1.5.

4.2. Logical centralization

A logically centralized control plane is a departure from traditional network protocols which are mainly distributed. However, experience has shown that some traffic engineering problems, such as QoS [10] and adaptive load balancing [8, 11], can better be solved with a global view of the network and its policies. Many other aspects of networking can also benefit from global optimization.

This centralization means that the controller needs to be able to fully control all network devices within the policy domain. This also means that the network devices must offer APIs for the controller to derive the topology, and to implement the monitoring and control of network resources across multiple devices.

OpenFlow is implemented over TCP/IP, which enables the use of any network topology between the controller and the network device. OpenFlow offers the ability for the controller to receive and send packets on each switch port, this may be used to do topology discovery and emulate existing protocols. The bundle mechanism, introduced in version 1.4, allows to store and pre-validate requests on each switch, which enables better synchronisation of changes across multiple switches.

A physically centralized controller is not desirable from a reliability and performance standpoint. In most cases, the controller needs to have a distributed implementation, which makes its implementation more complex than having everything in a single instance. However, the distribution of

the control function can be decoupled from the actual network topology, and this enables tailoring the distribution architecture to the controller implementation and the deployment requirements [12]. The controller can fully define the distribution granularity, the function partitioning, the data replication scheme and the consistency choices.

Significant work has been done to support multiple controllers in the OpenFlow specification (see Table 1). The role election mechanism, introduce in version 1.2, helps controllers to coordinate the control of a network device. Rich event mechanisms introduced in version 1.3 and 1.4 enable controllers to monitor and synchronise with the actions of other controllers in a fine grained manner.

4.3. Northbound APIs

The centralization of the control plane offers an easier way to expose an API from the network to non-network applications and middleware, and such an API is commonly called the NorthBound API [13]. In existing networks, the network infrastructure and the applications do not have a good way to exchange any relevant information. Applications and network devices cannot communicate directly, for example by adding metadata to packets, because usually applications cannot be trusted. The end result is that often the network device has to reverse engineer the intent of the application (for example using DPI), and the application has to infer how to use the network (for example probing for the network bandwidth).

The Northbound API defines a central place in the infrastructure where the global application policies and the network policies can be explicitly communicated and mediated (see Section 3.2). The NorthBound API is mostly independent of the OpenFlow protocol, as the two APIs are at a different level of abstraction. However, all the elements offered by the NorthBound API will need to be translated to OpenFlow features, and OpenFlow must support those application requirements. For example, the NorthBound API of the QoS controller module used for the SDN Lync demonstration exposed network flow latency [10], and this required specification improvements (see Table 1). The flow latency of the NorthBound API maps to the queue selection and rate limiting APIs in OpenFlow, introduced respectively in version 1.0 and 1.3.

4.4. Programmability

The need for real-time adaptation of the network means that SDN APIs must be designed for programs, not humans. Programmability enables automation software that can react and reprogram the network without involving humans in the critical path. Previous interfaces to network devices were mostly designed for human interaction (the CLI) or narrow management functions (SNMP: Simple Network Management Protocol). It is of course possible to program network devices using existing interfaces, but a well designed framework makes programming easier and enables a richer ecosystem.

The SDN framework must be flexible enough to handle all kinds of network devices, rather than one API per device type. This means dealing with both hardware and software devices, simple forwarding devices and devices with rich and complex behaviour. Having a common framework means that programs which are complex to build can be more easily repurposed to a different context. This

also means that a lot of developments and management tools, such as inspection and debugging tools, can be common.

The API must be designed as a whole and be consistent. The API should contain as few concepts as possible, and reuse the same techniques and objects. The features should be easily composable. The API should enable a complete feedback loop, and needs to integrate visibility and control together, so that programming can really be adaptive. Visibility has historically been more challenging than control, but programs can only react and adapt to things that are seen.

The programming paradigm of OpenFlow is based on flow tables and actions. There are no specialized processing structures, and almost everything is done in a flow table using actions. Group processing introduced in version 1.1 uses the same actions as in flow tables. The definition of header fields is common for matching them and setting them since version 1.2. The vast majority of structures defined in OpenFlow have statistics counters to increase visibility.

4.5. Flow entries

Packet forwarding defined by previous standards has mostly been coarse and tied to the protocol definition. For example, Ethernet defines forwarding based on destination address (and optionally VLAN ID), and IP defined forwarding based on destination address with subnet aggregation (i.e. using subnet masks). Only supporting these forms of forwarding is limiting. For example, tenant segregation and application QoS need forwarding based on other attributes. Changing forwarding behaviour with existing protocols is slow and complex because it must be standardized in bodies such as the IETF beforehand and adopted by most vendors before it can be deployed.

A key principle of network design has been the separation of network layers, where the operation of each layer is done without using information from other layers. However, more and more products violate those assumptions and now operate in a cross-layer manner. For example, many L2 switches have ACL based on the combination of MAC and IP addresses to prevent IP and MAC spoofing.

The OpenFlow API needs to offer flexible control and visibility of packet processing which is decoupled from the protocol definitions. The API should enable the collapsing of network layers as needed. Packet processing should be enabled at any granularity desired, as fine or as coarse as desired and suitable for the deployment.

The solution adopted by OpenFlow was the concept of a flow entry. The flow entry match describes a pattern of header values, though some header fields may be omitted (wildcard) or bit-masked. Flow entries enable to select related packets with flexible granularity and across protocol layers, and unrelated flow patterns can be used. In OpenFlow, most processing is attached to flow entries, and therefore flow entries are one of the most important concepts of the API.

5. Inside the Extensibility WG

The OpenFlow protocol was designed to address the evolution and extension of the SDN elements presented in the previous section in a generic and flexible way.

5.1. API and protocol

Most well-known network protocols are defined as peer to peer, as an information exchange between two network devices at the same level. However, OpenFlow is different in that it is defined in the control plane, as an API between one controller and one datapath.

Because it is an API, the specification of OpenFlow is more similar to the specification of APIs in various software frameworks, such as the specification of the Java API. OpenFlow only specifies the behavior of the switch, and the controller may use the API in any arbitrary fashion. All devices will not be able to implement all features, and so basic capabilities were added to enable a controller to discover the level of API support, and work is ongoing on profiles in the Forwarding Abstraction WG [14].

5.2. Flexibility guidelines

When the Extensibility WG was formed in the ONF, a few flexibility guidelines were decided to help the evolution of the OpenFlow API.

The first guideline is modularity. The core of the protocol should be as small as possible, and all the features should be simple and composable. Complex use cases should be broken down into simple elements that can be shared between use cases. Any part related to specific dataplane protocols should be kept outside the core of the specification. For example, in version 1.2, the definition of header fields was moved out of the core of the specification.

The second guideline is extensibility. Each feature should be defined in a generic fashion without arbitrary limitations, so that it can apply to a wide range of implementations and use cases. In each release, various features have migrated from a static encoding to more flexible encoding. The Experimenter Extension mechanism enable anybody to extend some of the specification objects structures (messages, matches, actions, errors) and is extended in each release. With each release, the Experimenter Extension API is extended and new object types can be defined. For example, in version 1.2, the specification converted the static match structure into defining match fields using the flexible OXM TLVs, and added a mechanism for Experimenter match fields.

The third guideline is interoperability. Any controller should work with any network device. Every feature must be fully specified, and features that are either opaque or switch dependant should be minimised. Network devices have different capabilities, capability discovery enables to query which features are supported and the size of the resources. For example, in version 1.3, the expression of table capabilities was greatly improved. After the release of version 1.3, this capability work has mostly transitioned to the Forwarding Abstraction WG [14], which developed the concept of Table Type Pattern (TTP) to better express switch capabilities [14].

The fourth guideline is consistency. The same convention, encoding and object structures should be reused through the specification whenever possible. This helps code reuse and developer familiarity. For example, in version 1.4, the property extensibility mechanism was better standardised and the queue feature was migrated to a standard encoding (multipart and properties).

5.3. Hardware and software

Hardware based and software based network devices have historically been developed independently, and even if they share the same overall functionality they have very different APIs. OpenFlow is one of the few APIs that deliberately tries to target both.

Hardware based devices usually trade-off limited flexibility for greater performance. They are usually based on ASICs, which have long development cycles, and this requires protocol stability. Software based devices are based on generic processors, are usually slower but much more flexible. New features can be deployed in matter of days, which enables rapid innovation.

The specification process in Extensibility is faster and more aggressive than a typical network specification process to try to address the needs of software based devices and to have a natural feedback loop between software implementations and the specification. At the same time, the review process evaluates the impact on hardware implementation of every change to the specification. The two main goals is to make sure the functionality of existing hardware can be expressed in the proposed API and that future hardware can take advantage of the richer API. For example, the set-field action introduced in version 1.2 allows software implementation to rewrite any header fields, and allow hardware devices to map header fields that they can rewrite.

Another consequence of the tension between hardware and software is that we maintain two branches of the specification. The main branch of the specification (versions 1.4, 1.5...) sees frequent releases to address software devices and is where new features are developed. In parallel, a long term branch has been designated (version 1.3.X); this branch is maintained for the benefit of the hardware implementations and includes only obvious bug fixes and clarifications, and no new features.

5.4. A Darwinian approach

The Extensibility WG is entirely composed of volunteers from different companies deploying or implementing OpenFlow. These people mostly work on topics of interest to their companies and themselves, and it is not possible to implement a top down process dictating the direction of development. The content of the specification is defined by those who do the work, and therefore, like in many other specification processes or open source software projects, it is impossible to fully predict future changes to the OpenFlow protocol.

New versions of the OpenFlow specification are released fairly often and include many new features and improvements, giving the impression that any feature proposed to the Extensibility WG is simply rubber stamped into the specification. In reality, the number of features proposed to the Extensibility WG that are not included in the specification is significantly larger than those included.

The initial development pace of OpenFlow was high, because OpenFlow was small and there were a lot of low hanging fruits to be picked, but as OpenFlow tries to address more complex areas, things have slowed down. The approach for managing proposals is very similar to open source projects, many proposals are worked on in parallel and only proposals that survive the incubation and prototyping review process are included in the specification (see Section 2). Very extensive changes are

considered, but obviously for those consensus is harder. In practice, the most likely proposals to be accepted are those derived from open source implementations and have already been deployed.

There are many reasons for proposals to be rejected. The specification process is designed to challenge proposals. The main reason for not including a proposal is lack of interest, where there are not enough volunteers to do the work to incubate and prototype the proposal. Many proposals fall into that category, such as TRILL framing support and MTU check support. Lack of consensus, where the technical experts don't agree on the proposal is another reason for proposals not making it all the way. This does not happen that often, as in most cases we eventually reach a consensus. One example of area where no consensus was reached is going backward in the pipeline, either via a goto-table going to an earlier table or a recirculation action. Some proposals are also not considered because they don't fit the spirit of OpenFlow and SDN, they either conflict with the SDN elements or the flexibility goals. Some proposals are not included for lack of review, where there is not enough experts to perform proper incubation of the proposal and evaluate its technical merits. One notable example is the color aware meter proposal extending the basic meters, which was proposed and not included in versions 1.3 and 1.4, and which is still considered for 1.5.

Finally, like in any specification process, sometime errors slip through the process and proposals included in the specification have bugs. The WG is very aggressive in addressing those errors, they are fixed in depth, and because of the frequent releases, bugs are fixed fairly quickly. One feature was even completely removed from the specification; the emergency flow cache was added in version 0.9, and removed in versions 1.1 and 1.0.1.

5.5. Roadmap for 2014

In 2014, the Extensibility WG will work on the following deliverable.

OpenFlow 1.3.X. This is the long term support branch for the specification mostly for hardware devices. It includes only clarification and obvious bug fixes (no new features). A significant amount of work is dedicated to process the feedback from vendors and plug fests, in order to make the 1.3.X branch correct and unambiguous. The release schedule is based on needs, it has been approximately one release every 6 months, version 1.3.4 was released in April 2014, work on 1.3.5 has not started.

OpenFlow 1.X. This is the main branch for the specification that mostly tracks the evolution of software implementations. It includes the work on making the API more flexible and adding new features. Version 1.4 was released in September 2013, version 1.5 is being worked on and planned for the second half of 2014, and version 1.6 would start after 1.5 is done.

ONF Extensions for 1.3.X. Many new features available in versions 1.X will be made available to 1.3.X implementations in the form of ONF Extensions. Those ONF Extensions use the Experimenter Extension mechanism in OpenFlow and can be bolted to the side of an implementation of 1.3.X without impacting its functionality. In combination with the long term branch, this allows hardware implementation to evolve and optionally pick and choose new OpenFlow features without impacting their compatibility with 1.3.X.

5.6. What's next

The next version of the stable branch, version 1.3.5, will contain only clarification and obvious bug fixes resulting from vendor feedback and plug-fests experience. Work on 1.3.5 has not started.

The next version of the main branch, version 1.5, is being worked on and planned for late 2014. The set of features in 1.5 is not finalised, some of the features may not survive the specification process. Some example features planned for 1.5 are delegation of learning, egress table, flexible statistics, matching on TCP flags, a copy-field action and support for optical switches.

Work has not yet started on the subsequent version, version 1.6. When 1.5 is done, any ONF member will be able to make proposals for 1.6 and push them through the specification process. Some people in Extensibility have expressed the desired to work on one of the following 5 themes: tunnel handling, delegation to the switch, L4/L7 and Network Function Virtualization (NFV) support, error handling and QoS support. Version 1.6 will most likely also include features developed by other workgroups in ONF. This may include for example support for wireless LANs and protocol independent packet processing.

6. Conclusion

SDN is designed to address networking needs that are poorly addressed by existing networks, and therefore the OpenFlow protocol looks more like a tightly coupled API. The evolution of the OpenFlow API is guided by a large number of unrelated use cases, the main elements of SDN and classical guidelines of API development. The goal of addressing a wide range of network devices, hardware and software, and the volunteer process in the Extensibility WG make the specification process quite complex to manage. To address these challenges, the specification process for OpenFlow was made very dynamic and quite similar to the process of open source projects.

Like most open source projects, OpenFlow is defined by its users and its implementers, and the Extensibility WG is a melting pot of ideas that need to be distilled into the OpenFlow protocol. From what has been accomplished in such a short time and the enthusiastic community, we know the future of OpenFlow will be exciting.

7. References

1. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communications Review, Apr. 2008.
2. IETF Forwarding and Control Plane Element Separation (FORCES) Working Group. http://datatracker.ietf.org/wg/forces/.
3. Nick Feamster, Jennifer Rexford and Ellen Zegura. The Road to SDN: An Intellectual History of Programmable Networks. ACM Queue, December 30, 2013.The Extensibility Working Group. https://www.opennetworking.org/working-groups/extensibility.

4. OpenStack Cloud Software. https://www.openstack.org/
5. Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. Extending networking into the virtualization layer. In Proc. HotNets, Oct. 2009.
6. Justin Pettit & Mark Pearson. Elephant Flow Mitigation via Virtual-Physical Communication. https://blogs.vmware.com/networkvirtualization/2014/02/elephant-flow-mitigation.html
7. SDN in Warehouse Scale Datacenters v2.0. Igor Gashinsky. http://www.opennetsummit.org/archives/apr12/1030%20Tuesday%20Igor%20Gashinsky.pdf.
8. Mike Schlansker, Yoshio Turner, Jean Tourrilhes, Alan Karp. Ensemble Routing For Datacenter Networks. Proc. of ANCS 2010.
9. HP and Microsoft Demo OpenFlow-Lync Applications-optimized Network. http://www.nojitter.com/post/240153039/hp-and-microsoft-demo-openflowlync-applicationsoptimized-network.
10. Wonho Kim, Puneet Sharma, Jeongkeun Lee, Sujata Banerjee, Jean Tourrilhes, Sung-Ju Lee, and Praveen Yalagandula "Automated and Scalable QoS Control for Network Convergence" Proceedings of USENIX INM/WREN 2010, San Jose, CA, April 2010.
11. Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma and Sujata Banerjee. Devoflow: Scaling flow management for high-performance networks. In ACM SIGCOMM, 2011.
12. Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A distributed control platform for large-scale production networks. In OSDI, volume 10, pages 1–6, 2010.
13. ONF – Northbound Interfaces Working Group. https://www.opennetworking.org/working-groups/northbound-interfaces.
14. ONF - Forwarding Abstraction Working Group. https://www.opennetworking.org/working-groups/forwarding-abstractions.
15. ONF - Configuration and Management Working Group. https://www.opennetworking.org/working-groups/configuration-management.