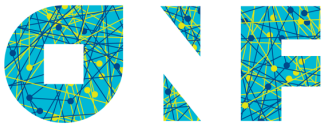
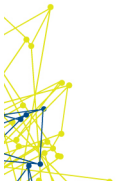


OPEN NETWORKING
FOUNDATION

Simplifying OpenFlow Interoperability with Table Type Patterns (TTP)

May 7, 2015



ONF Solution Brief
Simplifying OpenFlow Interoperability with Table Type Patterns (TTP)

Disclaimer

Table of Contents

List of Figures

1 Executive Summary

Open SDN has been well accepted by the networking industry as the way to transform Enterprise, Data Center, and Carrier networks. The objective of this SDN transformation is to simplify the network, lower total cost of ownership (TCO) and increase the speed of adding new services. The key attributes for a network migration to SDN are programmability, automation, and interoperability. Open SDN facilitates re-architecting the network to meet increasing demands imposed by rapid adoption of Cloud services and technologies.

ONF's OpenFlow (OF) specification has been successfully deployed over the past few years, primarily using OF1.0. This version exposes the abstraction of a single flow table in the switch to the controller. While single flow tables were a good starting point, they inherently suffered scaling issues when multiple switch functions were performed. With the advent of OF1.1, 1.2, and, most broadly supported, the anchor release of 1.3, support for multiple tables was introduced, allowing more flexible switch pipeline architectures. Multi-table support exposes a sequence of remotely controllable flow- tables, each supporting match-action capabilities that enable more sophisticated packet processing and services.

Several ONF Plugfest interoperability events have focused on OF1.3 switches from a given vendor interoperating with an SDN controller from a separate vendor. Such testing revealed that the flexibility of OpenFlow – having many optional elements, flexibility in number and order of applying actions to packets – makes achieving interoperability difficult. In other words, the immense variety of possible OpenFlow-controlled forwarding pipelines makes on-the-fly interoperability very difficult for the majority of today's switches.

Table Type Patterns, or TTPs, are an optional enhancement to the OpenFlow framework that allows controllers and switches to agree in advance on the forwarding pipeline details. The pipeline details spell out specific groups of OpenFlow rules (the "Pattern" part of the name "TTP") that will be supported by each Table in the pipeline. By creating TTPs for specific use cases, multi-vendor interoperability becomes simpler to achieve between switches and controllers sourced from multiple vendors. A key benefit of TTPs is that they enable SDN controllers to use OpenFlow to direct the operations of Legacy-Optimized ASIC-based switches, thus allowing SDN operation of already existing equipment, which is particularly important for near term adoption. This benefit derives from the pre-defined aspect of TTPs. In contrast to the "on-the-fly" nature of early OpenFlow, the define-in-advance nature of the TTP framework provides the development community with the ability to identify the TTP pipeline representations that can be mapped onto ASIC pipelines.

2 Interoperability Challenges with SDN

2.1 OpenFlow Multi-Table Support Offers Benefits . . . but Introduces Complexity

In OF1.0, packet forwarding within an OpenFlow-enabled switch is controlled by a single flow table that stores a set of flow entries managed by the controller. By manipulating the flow entries of switches, the controller may program the network as it determines the detailed forwarding behavior of each OF switch. Each flow entry contains match fields, action fields and a priority

assigned by the controller. The switch checks incoming packets against the flow entry match criteria. The highest priority matching flow entry defines (via its action fields) how to handle matching packets, e.g., forward, drop, mirror, etc...[1] The set of packets that match a particular flow entry (and no higher priority entries) is, by definition, a unique flow.

In OF 1.0, which supports only one controllable table, all matching and switch processing for a flow is described in a single table entry. While the single table model is simple and adequate for many control functions, it is too restrictive to address rich policy-based control functions that realize the potential for SDN architectures in a cost-effective and scalable manner.

Later versions of OpenFlow expanded to a sequence of multiple flow tables to enable more sophisticated and scalable processing, as illustrated in Figure 1, where a multi-table pipeline is described for bridging and routing, where each flow-table in the pipeline is used for a given purpose.

Switch processing use cases that would benefit from multi-table support include MAC Address Learning and Reverse Path Forwarding. While in theory both functions could be performed with a single, two-address lookup, this approach would result in an explosion of flow table entries and be prohibitively expensive in practical terms. In fact, this was an argument against OpenFlow when version 1.0 was released.

Another case where the pipeline may be optimized with multiple tables is with packet pre-processing. By pre-classifying Port-based VLAN IDs and Virtual Routing and Forwarding instances (VRFs), processing performance may be streamlined. While pre-classification could be implemented in a single flow table, doing so would be cumbersome and result in an explosion of flow entries. By contrast, pre-classification can be readily handled in a straightforward way by adding more flow tables, where the pre-processing is done by some flow-tables, and later flow-tables use the results and apply the main functionality.

While extremely powerful, multi-table OF pipelines expose performance variations among switches developed by multiple vendors, which may complicate the implementation of rich control functions. Because OpenFlow, by design, does not specify implementation choices, implementations can and do vary widely. For instance, a software-based pipeline vs. a highly optimized ASIC flow processor will offer dramatically different pipelines to be controlled.

Furthermore, controllers do not have a good way of assessing the functional differences of the switches they seek to control, such as the optional features supported. Consequently, controllers may be constrained in delivering network functions, such as application of fine-grained policies, because of the complexity of coping with significant variations in switch capabilities and performance. This may result in a ‘Least Common Denominator’ approach, where functionality is governed by the least capable switch. That, in turn, may preclude fully realizing the benefits of SDN.

Since the goal of SDN is to optimize the network around the needs of diverse applications, rather than vice-versa, additional mechanisms are required to exploit high-performance and highly capable implementations while avoiding requesting capabilities that a switch cannot fulfill. If

[1] In theory a packet could match two different flow entries of the same priority. Because OpenFlow1.0 declares the resulting behavior to be “undefined”, well-designed applications and controllers must avoid creating flow entries where that might happen.

controllers are aware of the capabilities of the switches they are controlling, they could optimize flow entry processing for each individual implementation.

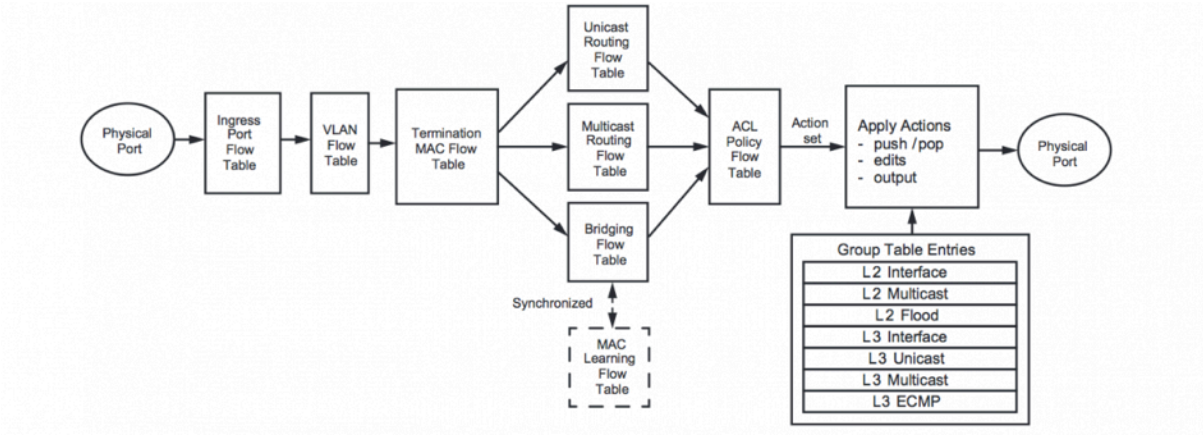


Figure 2.1: Abstract Switch Pipeline for Bridging and Routing

OpenFlow and Pipelines:

OpenFlow versions 1.3 and above implicitly specify a forwarding pipeline in the underlying network element, by allowing a single packet to be processed by multiple tables as it traverses a switch. In the absence of the TTP framework, an OF1.3 pipeline is incrementally defined by "**FlowMod**" messages that describe the behavior of each table. Incremental definition works well for software switches which can have as many tables as needed, each with whatever capabilities are required, but this approach does not scale for hardware-based switching implementation with finite resources.

In ASIC-based switches, the implicit pipeline that is outlined "on-the-fly" by a series of "**FlowMod**" messages is unlikely to match the capabilities of the switch unless it has been set up using prior knowledge. In other words, the controller-switch interoperability for OF1.3 cannot be guaranteed in multi-table scenarios without assuming the applications or controller writers having prior knowledge of the switch implementation. In short, this model fails to deliver the full marketplace decoupling of control and data planes.

A TTP explicitly describes a logical forwarding pipeline in OpenFlow terms. Switches, even mainstream ASIC-based switches, can typically support a variety of logical forwarding pipelines, typically one complex pipeline and other simpler pipelines that are subsets of the complex pipeline. Because each supported pipeline can be described by a different TTP, switches are able to support multiple TTPs (though only one can be active at any time).

Similarly, a given SDN use case may be supportable by more than one pipeline, so an application or a controller targeting a particular use case can interoperate with several different TTPs. This flexibility on both ends enables a dynamic model whereby TTPs (which are required to have unique identifiers) are negotiated and agreed between a switch and controller at connection time,

providing both sides with the context about what is needed and available in terms of forwarding behaviors.

TTPs are required to have unique names. Switches and controllers can then explicitly advertise support for given TTPs, easing compatibility and interoperability.

2.2 The Answer- Table Type Patterns (TTPs)

Table Type Patterns (TTPs) provide a method for determining, before live traffic flows, either a priori or negotiated, the end-to-end flow processing behavior within an SDN. This has tremendous benefits to SDN in terms of consistency and predictability. Since SDN must deliver consistent behavior across heterogeneous and distributed elements, a method is needed to align the flow processing capabilities of all switches and controllers with the requirements of any given network application that will push flow rules into the network.

TTPs fully describe switch behavior based on the OpenFlow 1.x multi-table switch model and provide an abstraction of the OpenFlow flow processing capabilities of the underlying physical and virtual switches. By mapping the unique flow processing pipeline of any given underlying switch to a specific set of flow tables and valid entries for the flow and group tables in an OpenFlow logical switch, a consistent set of forwarding behavior can be formally specified in advance. This specification can be rigorously analyzed, simulated and otherwise validated, to ensure robustness. This in turn allows a network operator to deploy a TTP-based SDN implementation with confidence that the various elements will work as advertised.

Just as the x86 instruction set has provided an abstraction for a common set of capabilities for different underlying physical hardware platforms upon which a rich set of interoperable, higher-level software has been developed, TTPs provide a similar approach to OpenFlow-based SDN. A TTP can be initiated from any point in the ecosystem – for example, ONF or other SDN community, chip or switch vendors, application developers or SDN architects.

The diagram below is helpful in illustrating the expected lifecycle of a TTP. Ultimately an end-user must highlight a set of capabilities needed for network control and motivate an SDN architect (or SDN application developer) to define a TTP that satisfies the application's needs. Alternatively, chip or switch vendors may develop a TTP to expose capabilities in their devices that is readily accessible in a common way to network application developers.

TTPs can be developed by any member of the OpenFlow community, a vendor, a network operator or a group with shared interests such as an ONF working group. The purpose of TTPs is that they be shared to provide a common understanding of the logical (abstract) pipeline that will be controlled by OpenFlow in a given context. It is worth clarifying that sharing can be public (in the full spirit of SDN) or private (i.e. under NDA), which may be appropriate during early development or when a network operator wishes to share requirements with a vendor.

TTPs are an optional enhancement to the OpenFlow switch framework in order to achieve multi-vendor interoperability. While they may not be needed in all deployments, they formally align switch capabilities with network application and controller requirements that may be adopted across an open ecosystem.

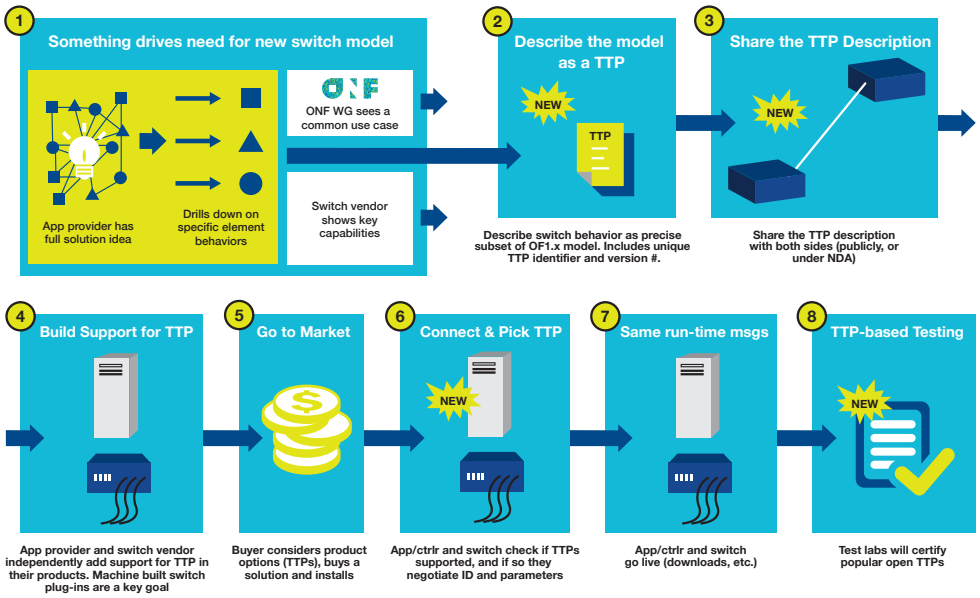


Figure 2.2: Life Cycle Overview for a TTP

TTPs are the first generation of a broader framework called Negotiable Datapath Models (NDMs). TTPs are compatible with OF 1.x versions that provide a way for network applications, controllers, switches and silicon to agree on pipeline models for end-to-end flow processing. As the OpenFlow protocol evolves, future versions of NDMs are expected to emerge which provide more extensive functionality.

2.3 OpenFlow Implementation Challenges

As described in the previous section, TTPs use OF 1.X constructs to describe specific forwarding pipelines unambiguously. Those specific forwarding pipelines are created to deliver specific controllable data plane functions such as VxLAN gateway, L3-router, etc.

TTPs address multiple challenges including the following:

1. **Interoperability Complexity**
   The OFS specification is aimed at a superset of data plane forwarding behaviors with many mandatory and optional elements. With many vendors developing switches and controllers for a broad range of applications, implementations are often dramatically different. As a result, validating interoperability is extremely challenging and expensive.

   From an SDN network application's perspective, the required level of interoperability is achieved when switches can be programmed to achieve the desired behavior. Additional functionality enables the switch to increase its value by addressing an even broader set of use cases.

By defining subsets of required functionality for specific use cases, TTPs enable controllers to efficiently assess which switches are capable of supporting the required forwarding behavior in response to network application demands. Over time, we expect that some TTPs will be more broadly adopted than others. Broader adoption will motivate convergence toward a small number of TTPs (potentially just one) that address specifically targeted use cases, simplifying the quest for interoperability.

2. **Data Plane Implementation with Multiple Tables**
   Versions of OpenFlow after OF1.1, support multiple flow tables to enable data plane switches to address complex use-cases. Unfortunately, mapping of multiple flow tables into existing fixed function ASIC-based switch platforms becomes cumbersome and sometimes may not be possible with on-the-fly OpenFlow.

   Writing an OpenFlow agent that can handle arbitrary pipeline processing becomes an unnecessarily complex exercise, especially when the use-case is known. The expectation that OF forwarding behavior (pipeline processing) can be changed arbitrarily at run-time (on the fly), introduces unwarranted complexity in the implementation. In addition, on the fly interactions between elements (such as controllers and switches) may induce non-deterministic behavior (bugs) in the field. The on-the-fly approach is much harder to validate than a prepared model. To gain the trust of network operators, on-the-fly implementations will need to be thoroughly tested across relevant vendor options. This means extra work and effectively undermines any perceived benefits of on-the-fly functionality.

   TTPs define clear forwarding behavior in OpenFlow 1.x terms for specific use-cases (i.e. specific forwarding functions). These advance definitions enable deterministic behavior that can be pre-validated in order to minimize field issues and ensure network reliability. By reducing implementation complexity, vendors can reduce switch testing and TTM without any loss of trust on the part of network operators.

3. **Data Plane Resource Utilization**
   TTP-based implementations, by spelling out the needed switch functionality, can allow switch software to optimize run-time resources relative to unspecified (full-featured) OF implementations. Use-case specific optimization improves scalability (e.g. flow/connection /tunnel capacity) and performance (e.g. throughput and latency) without compromising flexibility. Major vendors are already beginning to incorporate TTP support into their products that offers run-time optimization for multi-table pipelines, with many more expected over time.

4. **OpenFlow Adoption**
   OpenFlow adoption requires an open ecosystem of users (network operators, service providers and enterprises), vendors (of applications, controllers and switches), testing houses and standards bodies such as ONF. TTPs achieve the promise of SDN/OpenFlow interoperability through incremental investment, use case by use case, with each interest group empowered to make progress as desired. An incremental approach reduces risks, enables migration, and allows organizations to pace their individual transition to SDN.

Switch vendors can implement optimized, purpose-built, OpenFlow-enabled devices that can be pre-validated and certified to operate with controllers and network applications.

Similarly, end-users and operators may increase their confidence levels that OpenFlow switches and controllers will interoperate and be extensible as their needs evolve. TTPs are a key enabler to expedite OpenFlow adoption with incremental effort and resources, rather than forcing investment in flexible, yet complex, platforms that are excessive, given their requirements. Ultimately, it will be the broad adoption of SDN that provides the necessary economic clarity for chip innovators to make multi-million dollar investments.

5. **Optimizing Interoperability Testing**
   TTPs can serve as test profiles for interoperability testing that reduce the time, effort, and cost to validate specific applications. This is especially important considering the rich set of features that OpenFlow provides.

3 Additional TTP Benefits

3.1 Product Compatibility

Take the diversity of OpenFlow switching hardware pipeline implementations, cross this with the variety of SDN applications, controllers and use cases, factor in the various OpenFlow releases and it quickly becomes difficult for buyers, vendors, and partners to determine which products are readily compatible with one another. Operators could either expend great effort to perform interoperability testing, in a generic fashion, or they can alternatively adopt TTPs.

TTPs simplify the product compatibility matrix by allowing suppliers to document their OpenFlow support in a common and straightforward way. Apps and controllers that are designed around TTPs will constrain their OpenFlow messaging to the boundaries of the models. Switches that support TTPs promise to deliver forwarding services according to those models. When both sides support the same TTP, compatibility is predictable. This makes it clear to the buyer which use cases could be supported. For suppliers and partners, it removes confusion and promotes development of a robust ecosystem of SDN solutions because different elements of the system are communicating with a common understanding of the underlying pipeline.

3.2 Procurement and Upgrade Clarity

Much SDN and OpenFlow discussion has been focused on early use cases. Now, with the widening appeal of SDN OpenFlow, network architects are looking for broader deployments, upgrades and multiple sources of supply.

TTPs, with their unique identifiers, can be used by these architects to define the specific needs of their networks. Whether early in the architecture (perhaps as part of an RFI) or further along, TTPs allow vendors to respond to clear requirements and to provide unambiguous responses, as defined by the TTP, while offering architects the ability to compare “apples to apples” and assemble multiple solutions that meet their defined needs.

TTPs also ensure smooth network upgrades by clearly outlining new device capabilities so there is compatibility with other devices on the network. TTPs create a common communication language, and level of cooperation, between application and network support to make network changes seamless.

3.3 Test Profiles and Performance Checks

TTPs can be used as a proven set of test profiles to validate OpenFlow's mandatory and optional functions. Scalability and performance benchmark profiles can also be represented as TTPs. Defining scalability and performance tests in common terms such as TTPs, market participants can compare results in an apples-to-apples way, building trust which will help advance adoption.

Additionally, an industry standard TTP test profile can also be established to ensure 'minimum functionality' screening. TTP test profiles allow vendors to claim levels of functionality that are meaningful and understood by other vendors and buyers, mitigating the risks that existed for the initial products.

4 Conclusion

TTPs are a method to solve the OpenFlow interoperability challenge when dealing with different switch pipeline implementations, whether a hardware or software switch. ONF is working together with switch and chip manufacturers to define a set of TTPs that can be supported across many product lines. This is a step in the direction of defining application profiles that can be implemented end to end in a network, for true deterministic application performance.

For more information on the technical details of TTPs please see the reference documents in Section 5.

5 References

TS_OpenFlow_Table_Type_Patterns_v.1.0_052014.pdf (June 04, 2014)

TS_OpenFlow_NDM_Synchronization_v.1.0_042014.pdf (June 09, 2014)

TTP FAQ dev page (June 18, 2014)

Broadcom OF-DPA version 2: https://github.com/Broadcom-Switch/of-dpa/tree/master/OF-DPA-2.0

TTP Project at OpenDaylight: https://wiki.opendaylight.org/view/Table_Type_Patterns:Main

6 Contributors

Nabil Damouny

Fahd Abidi

Daniel Williams

Komer Poosari

Carolyn Raab

Curt Beckman

Shaji Nathan

Doug Marschke

Michael Orr