Charter: Forwarding Abstractions Working Group

INTRODUCTION

Over the past several decades, the “Moore’s Law” effect has seen network hardware improve along almost all hardware oriented dimensions, including port densities, speeds and feeds, and price per bit. At the same time, network control plane evolution has been dramatically slower. Configuring and managing networks continues to be about network element operations rather than network services. This mix of ever-growing networks and increasingly complex configuration requirements makes network agility, service velocity, operation, and debugging increasingly difficult and expensive. This effect is seen across all network types, including enterprise, data center, and service provider networks. The OpenFlow™ Standard and software-defined networks are widely seen as comprising a promising set of solutions for resolving these challenges.

However, even with the spurt of innovation that we’ve seen around the OpenFlow protocol and related architectures, adoption of the OpenFlow protocol has proceeded at a frustratingly slow rate. The rate of adoption on hardware forwarding targets such as ASICs, NPU, and Network flow processors has been even more problematic. In addition, recent versions of the OpenFlow Standard, the so-called 1.1+ versions, have seen even less adoption.

Several efforts have been undertaken to analyze the OpenFlow Standard framework in the context of implementation on hardware forwarding targets. These efforts have identified a set of problems with the framework, including:

- Information Loss (e.g., Table Typing and Table Relationships)
- Information Leakage (e.g., requiring a controller to incorporate target specific code) Weak Control Plane/Data Plane Interface (e.g., Pipeline Order)
- Combinatorial State Explosion
- Weak Indirection Support
- Limited Expressibility or Network Primitives (e.g., Data Plane, Replication, Parse/Rewrite)
- Run-time functional mapping, which makes interoperability difficult to determine in advance

All OpenFlow versions up through 1.3 describe a framework in which controllers request switch forwarding behavior in a step-by-step fashion (“flowmod by flowmod”) at run-time. In this framework, the platform Hardware Abstraction Layer (HAL) must be capable of mapping individual flowmods on-the-fly to the underlying platform1. This framework necessarily describes specific low-level actions that detail how the network device must behave rather than describing what switch-level behavior is required to implement the network application. By over-specifying the details, and by forcing run-time HAL-based mapping of actions, OpenFlow has created an unnecessarily difficult implementation problem for less-flexible target HALs. This difficulty adds to adoption resistance.

The Forwarding Abstractions Working Group (FAWG) proposes to enable pre-run-time description of switch-level behavioral abstraction to address these two issues. By describing switch-level behavior in advance of run-time, HAL developers have enough information to know which details of their platform’s low-level behavior (the “how”) is relevant or immaterial to the larger required behavior (the “what”). In addition, HAL developers will be relieved of the requirement to implement complex mapping functions

1. Note that in some cases it will not be possible to map a sequence of flowmods or tablemods to a hardware forwarding target. A canonical example is shared tables, in which the HAL would, at run time, have to decompile the intent of the controller writer to discover that two tables were actually an instance of one table with multiple views.
(bordering on Hardware Description Language compilation) in their HALs. Note that there is an additional market-supporting benefit that comes from using negotiated identified abstractions: when a controller and a switch support the same abstractions, interoperability is assured. This assurance will become stronger when compliance testing is added to the picture.

The FAWG proposes to develop the negotiated abstractions framework in two phases. This charter details the first phase, and includes the intentions of the second phase so that the first phase appropriately prepares for the later phase. Phase 1 focuses on easing the challenge of mapping behavioral descriptions expressed in terms of Match Action Tables onto existing hardware by providing a means for sharing well-known “Table Type Patterns” in advance of run time. Phase 2 will go beyond the Match Action Table construct by providing a fully expressive set of primitives and logic. Richer expressibility will enable better solutions in the Ethernet/IP domain, and also prepare OpenFlow to address new domains, such as connection-oriented and other spaces. Both phases will bring maturity to OpenFlow, and increase alignment with the multi-billion dollar network hardware market.

Phase 1 assumes the same pipeline abstractions2 (e.g. Match Action Tables and flowmod messages) that currently exist in the OpenFlow framework. The OpenFlow switch model defined in OF1.1/1.2/1.3 describes a hugely flexible series of powerful “TCAM-like” tables, with more capability than current or announced hardware platform implements. The paradigm is that a controller uses some more-or-less modest subset of that rich switch model. However, at present, the switch does have any a priori knowledge of the subset that the controller intends to use. The negotiated abstractions to be developed in Phase 1 simply describe agreed constraints on that rich table-based switch model. No new switch behavioral capabilities are being added as part of this phase.

The benefits of this phased approach are 1) that transitioning to the new framework should be straightforward, 2) controllers and flexible (soft) switches can quite reasonably support the earlier (non-negotiated) framework in parallel with the new (negotiated abstraction) framework, and 3) the deliverables time frame for Phase 1 is much shorter than for Phase 2. The drawback is that Phase 1 cannot address fundamental problems of expressibility that result from the current table-based OpenFlow Standard switch model.

The key first phase deliverables of the FAWG establish what we call Table Type Patterns, or TTPs, as the negotiated switch-level behavioral abstraction. A TTP consists of the linkages between tables3, the types of tables in the graph, a set of the parameterized table properties for each table in the graph, the legal flowmods/tablemods for each table, and the meta-data+mask that can be passed between each table pair in the graph. This implies that a representation of the TTP will need to be developed and standardized. A representation of a TTP in set notation is included in an appendix to this document.

As mentioned above, Phase 1 is about recognizing that the controller normally only uses a (modest) constrained subset of the table-based OpenFlow Standard switch model, and that it would be helpful for the switch to know the constraints in advance. As we have looked into describing the constraints, we have recognized the need to specify some new table types beyond those listed in the Table Features messaging section of the existing (OF1.3) specification. At a minimum, we see that we need wildcard, exact match indexed, exact match keyed, and longest prefix match tables. In addition to the table types, the table

---

2. While we use the same abstractions (e.g., tables), they are constrained by functionality. For example, an exact match table can be seen as a constrained version of a wildcard table.

3. These linkages are expressed as a directed graph in which the nodes are tables and the edges are the connections between tables. See the description of TTPs.
typing capabilities in the current OpenFlow specification will need to be augmented, along with various other protocol capabilities such as OpenFlow Extended Match, table features, and specification of which flow modifications are legal for each table type. The FAWG will work with the Extensibility WG to resolve these and other specification extensions.

FAWG will work to specify the TTP description scheme in such a way that the scheme carries forward across versions of the OpenFlow Standard. That is to say, FAWG will be working in an initial context of OF1.3 with the intention that OF1.4 (and other versions) will be supported without special treatment. However, if OF1.4 begins to move forward with some unanticipated changes, FAWG will need to be alert to any necessary adjustments.

Scheduling information for Phase 1 deliverables and milestones is provided in the “Deliverables and Timelines” section below. The planned-for Phase 2 will address expressibility limitations in the current Match Action Table paradigm. More information on Phase 2 is provided in the “Anticipated Phase 2 Deliverables” section below.

WORKING GROUP NAME
The working group will need an appropriate name.4 We are proposing “Forwarding Abstractions Working Group” as the name, since both phases of the effort deal with defining forwarding abstractions. In Phase 1, the abstraction is the Table Typing Pattern. In Phase 2, the abstraction is the Forwarding Plane Model. However, there is no strong attachment to that particular name, and the group is perfectly willing to use any other name that the TAG or ONF Board may specify.

GOALS AND RESPONSIBILITIES
The Phase 1 goals for the FAWG are to:

• Enhance OpenFlow Standard adoption on hardware forwarding targets
• Ensure that adoption on software forwarding targets is not impacted
• Enhance testing and certification by establishing canonical Table Typing Patterns (TTPs)
• Ease controller implementation by encouraging broad adoption of common TTPs
• Simplify assessment of interoperability via adoption of a small set of TTP “code points”
• Provide for member-owned TTPs to enable market-driven convergence on code points
• Ensure that existing capability is still supported:
  • When no negotiated abstraction, works as before
  • In presence of negotiated abstraction, some changes legit

The responsibilities of the FAWG are to:

• Establish the WG and conduct regular calls with members
• Maintain the integrity of the WG to keep ONF goals the top priority
• Liaise with the other WGs to maintain coordination and clear delineation of responsibilities
• Develop the identified canonical Table Typing Pattern (TTP) prototypes

4. A frequent suggestion is that the group should continue to use the Future name, and simply convert the “Discussion Group” to a “Working Group.” There are two issues with this. First, assuming that the charter is accepted, the need for a Future Discussion Group will continue, and in that sense the name is “already spoken for.” Second, it’s useful to name deliverables after the group name, and names like “Future Spec X” or “Future White Paper Q” are problematic.

5. In addition, the Work Group will encourage both HAL and controller side Proof of Concept implementations.
• Work with members to ensure that the TTP framework is compelling to them (will be adopted)
• Encourage members to develop and share useful “proof of concept” implementations
• Maintain the mailing list, repositories, wiki, and other items related to the WG

See also the “Deliverables and Timeline” and the “Anticipated Phase 2 Deliverables” sections below.

PROPOSAL FOR CANONICAL TABLE TYPING PATTERNS
Several candidates for canonical TTPs are potentially interesting.

1. OpenFlow 1.0 Forwarding Element
2. Constrained OpenFlow 1.1 Forwarding Element
3. Layer 3 IPv4 Forwarding Element

PROPOSAL FOR COMMON FORWARDING PLANE MODELS
Each of the common Table Typing Patterns will be re-expressed as FPMODS models. In addition, several additional models will be developed (which seem difficult to address within the Table Typing framework) in Phase 2.

1. Stateless Generic Tunnel Encap/Decap Element
2. Stateful Generic Tunnel Encap/Decap Element
3. 802.1D Forwarding Switch Element
4. Layer 3 IPv6 Forwarding Element
5. Dual stack (v4 + v6) Forwarding Element

DELIVERABLES AND TIMELINE

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publish a TT Patterns architecture document</td>
<td>21 Sept 2012</td>
</tr>
<tr>
<td>Specify a canonical set of TT Patterns to be prototyped by the WG³</td>
<td>28 Sept 2012</td>
</tr>
<tr>
<td>Standardize the process for registering identifiers for TT Patterns</td>
<td>12 Oct 2012</td>
</tr>
<tr>
<td>Rough draft of TTP description methodology</td>
<td>12 Oct 2012</td>
</tr>
<tr>
<td>Rough draft prototypes of canonical TTPs</td>
<td>26 Oct 2012</td>
</tr>
<tr>
<td>Standardize a protocol for controllers and switches to negotiate TTP IDs and parameters</td>
<td>9 Nov 2012</td>
</tr>
<tr>
<td>Consensus on standards for new and/or extended features², including but not limited to table match extensions and capability negotiation</td>
<td>16 Nov 2012</td>
</tr>
<tr>
<td>Standardize the methodology for describing and representing TT Patterns</td>
<td>14 Dec 2012</td>
</tr>
<tr>
<td>Finalize the prototypes of the canonical TTPs</td>
<td>11 Jan 2013</td>
</tr>
<tr>
<td>Review of Phase 2 deliverables (below) and submission for ONF approval⁸</td>
<td>18 Jan 2013</td>
</tr>
</tbody>
</table>

6. The prototyped TT Patterns will be assigned IDs but will not be formally “standardized”. They will serve as examples and “test vehicles” for the development of the standards related to TT patterns.

7. The design and specification of OpenFlow extensions and/or new features will be done in consultation with other working groups.

8. Work on Phase 2 deliverables will require board-approved timeline for those deliverables
ANTICIPATED PHASE 2 DELIVERABLES

Some aspects of Phase 1 deliverables will be influenced by the Phase 2 goals. Consequently we are including the anticipated Phase 2 deliverables for reference.

Phase 2 Deliverables:

- Extend the “negotiated abstraction” approach of Phase 1 to support more flexible “models”
- Identify a handful of useful common FP models to be initiated by the workgroup
- Codify the layout and required elements of forward plane model descriptions
- Codify the means of creating primitives for use in constructing FP models descriptions
- Create a common library of primitives
- Document a straightforward process for adding new primitives as needs arise
- Construct FP models for and establish a clear migration path from 1.x to FP Models framework

CONSIDERATIONS

As this workgroup charter is being drafted, some tensions are foreseeable that will need to be resolved as part of the workgroup’s effort (i.e. they are too complex to resolve easily prior to establishing the WG, or they will involve working with other WGs). We list some here which bear consideration.

Operational Mode: This effort involves negotiating forwarding plane abstractions that include the notions of Table Types. There are potential interactions with the Table Features message capabilities. Because both of these mechanisms can define table types, only one should be active at a time. Effectively, this suggests that the controller-switch connection might have two possible operational modes: “pre-negotiated” and “dynamically-mapped”, at least with respect to Table Features messaging. Resolution of this tension will require working with the Extensibility Working Group.

Flowmod Enforcement: A TTP effectively describes a constrained OpenFlow switch model, which limits the number of tables used, and the actions and linkages allowed for each table. When a controller and a switch have negotiated an agreed TTP, which end should enforce the limitations implied by the TTP?

Redundant Goto Actions: When a TTP has been negotiated, then table linkage is known. For tables that always pass control to the same next table, goto actions are redundant, add no value. However, today if flowmod does specify goto processing terminates. In the presence of a negotiation TTP, should the goto semantics be modified to eliminate the redundant goto action, or should the semantics be kept as-is (or should both options be allowed, settled at negotiation time)?

Longest Prefix Match Table Type: “Longest Prefix Match” is an important table type that should be included in the TT patterns concept. However, this notion is not explicitly called out in current flowmods. There are multiple approaches to provide resolution of the potential challenges there. Other workgroups will be consulted in order to resolve this.

TTP Identifiers Version Fields: TTP identifiers must obviously be tightly tied to the table-typing pattern they represent. Having said that, TTPs will not exist in isolation; they will couple to use cases. As such, TTPs might need to be revised over time. Each revision should have a unique identifier, of course, and at the same time it should be easy to see that separate versions are related. In short, a version field should be part of the identifier scheme. This is true whether the ID is numerical (e.g. a 64-bit value) or XML-style.

Run-time parameterization: Table Typing Patterns will include a lot of attributes about the tables in the graph. It may be desirable to allow for run-time negotiation for certain table attributes. (Run-time parameterization will help allow a single TTP fit more targets while still allowing for exploitation of optional
features in switches that can support them.) For example, the total number of tables, how the tables are linked, what actions are allowed in each table, what match type is applicable for a table, etc. Table size is an interesting attribute but it’s unattractive to associate a TTP with specific table sizes because it immediately implies an explosion of TTPs which are identical in all ways except table size. So we need to resolve the tension around table size. TTP might describe minimum table size, or possibly a range of table sizes. If so, then there may be a need to negotiate specific table sizes at run-time. Conceivably other aspects of tables (e.g. optional actions for a specific table) could be “parameterized” and negotiated at run-time as well, although this complication may not be as compelling as allowing for negotiation of size.

APPENDIX: REPRESENTING A TTP IN SET NOTATION

A TTP can be thought of as the following set:

\[ TTP = \{ G, \Gamma, \Pi, \Psi, \Omega \} \]

where \( G = (T,E) \)

where \( T \) is a set of tables (vertices and \( E \) is a set of edges such that table \( Ti \) precedes table \( Tj \) iff \( \exists (E_i, E_j) \subset E \)

\( \Gamma : T \rightarrow \) the set of table types

\( \Pi : T \rightarrow \) the set of possible table parametrizations

\( \Psi : G \rightarrow \) the set of legal flowmods and tablmods for a table

\( \Omega = \{ \mu(i,j) \mid (Ti,Tj) \subset T \land \exists (E_i, E_j) \subset E \} \)

where \( \mu(i,j) \) is the meta-data that can be passed from table \( Ti \) to table \( Tj \)

(Note: Due to some technical challenges in converting this notation between document formats, the expression above was inserted as a screen shot. Please excuse some of the artifacts.)