Model-Driven OpenFlow Interoperability

Curt Beckmann
ONF ODWG Chair, ODL TTP project lead
Chief Technology Architect, EMEA, Brocade
Abstract

The SDN movement is based on abstracting the data plane to allow device-independent control. The 1.0 version of OpenFlow provided a basis for decoupling in simple situations. But establishing a common interoperable interface for device abstraction has proven more difficult for complex networking using the simple OpenFlow framework. A new richer model-driven framework that describes forwarding behavior in OpenFlow terms promises to bring practical device independence to complex networking. This talk will describe the framework, some challenges, and recent progress at the ONF and in OpenDaylight.
Comment

I first thought that EWSDN had an academic focus…

• Many industry challenges around SDN arose from implementation/deployment history

• Academics is not usually as focused on implementation and deployment problems…
  – but solving them, and solving future related issues, seems interesting and can make use of new ideas which begin in Academia
Sequence

• Background: Device Pipelines
• Framework: Table Type Patterns (TTPs) as Pipeline Models
• Architectural Iterations
• Header Space Analysis goes Mainstream?
• Q & A
Background:

Early OpenFlow

• SDN promise: open decoupling of control and data planes
  – “Open” meant vendor decoupling
  – OpenFlow introduced as “standard low level control protocol”
  – Many vendors offered OF-enabled boxes: Problem solved!
    • Just a few details to work out, which we will leave to others
    • Next trick: fully programmable data planes (another story, not for today)

• OF1.0 assumed a trivial pipeline: one match-action table
  – Good news: Supportable on many devices
  – Bad news: Too limiting
A Few Details

• OF1.0 limits noticed early
  – No complex handling, limited scale
• OF1.1 gave us 255 flow tables
  – Published pre-ONF… skipped the “working code” policy
• OF1.1 opens the door for complex forwarding pipelines
  – Allows for complex handling and scale
• But no recognition of the diversity of existing pipelines

Framework Gap

• The OF framework lets the controller to send any legal OF messages
  – Device must handle them… That only works with pipeline agreement
• Founders of Forwarding Abstractions WG* anticipated this challenge
  – Realized that OF needed a way to get pipeline agreement

* Now a part of the Open Datapath WG
Sample Pipelines

Both are from Broadcom’s OF-DPA…
See: https://github.com/Broadcom-Switch/of-dpa
Dealing with Pipelines

• Dealing with a variety of pipelines makes life harder.
  – If everything were an NPU, we could make all OF features “required”
  – Even “flexible ASICs” are envisioned, but will differ, can’t do “everything”
    • From market perspective, that has other challenges
      – Are the all identical? No innovation? Must we swap out all boxes in the world? A bit risky!
  – Anyway, we don’t have NPUs or programmable ASICs everywhere yet…
    • We get to choose whether to advocate a “flexible only” path forward…
      – Serious adoption challenges there!
    • Or see if we can create SDN-based value using existing “less flexible” ASICs of today

  – I am in the practical camp: “Let’s see what we can do today!”
    • SDN adoption on today’s platforms will simplify adoption on future platforms
Pipelines Models

• Original OpenFlow Pipeline was trivial and abstract
• The diversity of physical pipelines makes common code tricky
• But we can create common pipeline models (like previous slides)
• And then map those models onto physical pipelines
  – OpenFlow 1.0 did this with a common (but trivial) pipeline model
  – In OF1.0, that mapping was implicit, done at development time
  – Most devices had some table (usually “ACL table”) with a “TCAM”
    • The coders just mapped the (single) OpenFlow table onto the TCAM
Framework based on Pipeline Models

• After OF1.0, pipeline mapping got much harder
  – The OF pipeline model was no longer a subset of device pipelines
  – Now the OF pipeline is a superset of ASIC pipelines
    • Though most NPUs were capable of handling the OF pipeline model
  – But most ASICs do have interesting pipelines worth controlling

• So… How to enable control of existing ASIC pipelines?
  – Run-time mapping of multi-table OpenFlow messages way too hard
    • Arguably impossible… not enough information available, etc
    • Mapping problem much to hard to solve dynamically in a switch… outcome uncertain
    • Network operators don’t want uncertainty at run-time… sort that out in advance
  – Proposal: Figure out the mapping before run-time!
First Framework Approach

• Our early approach was very switch oriented
  – Most participants were device vendors
  – There were many controllers out there, not represented

• Early picture was “controller tells switch…”
  – Top-down “Use Case” pipeline models seen as Primary
Early Approach

Table Type Patterns (TTPs) were developed by the ONF’s ODWG*

Early approach envisioned switch vendors adding numerous TTP agents to their devices. Possible, but “uphill” given that TTPs will iterate at first, and switches are not agile development environments.
Some of the early challenges…
That we didn’t see at the time

• Model to Physical mapping was based in switches, a bit weak
  – Switches are not great places for code development
  – Switch vendors tend to release code only occasionally (6 to 9 months)
    • Early pipeline models (TTPs) will need to iterate a few times quickly
  – Switches do not typically host 3rd party code

• But as mentioned, we didn’t realize these challenges at the time
Next Step: Express pipelines in OpenFlow

• In charter phase, WG told not to change OF protocol (!)
  – OpenFlow Switch protocol specifies control messages

• We had to “work backwards”... express pipeline models as allowed/supported messages per pipeline stage (Flow Table)
  – We expected that many common usages for OpenFlow would end up generating certain common patterns of messages for certain tables...
    That is where the term “Table Type Patterns” emerged.
  – We see a TTP as a set of constraints on “OpenFlow message space”
  – When a controller
    • The controller can only send messages within that OpenFlow message space
    • The switch must support messages in that OpenFlow message space
Challenge: How to express Constraints

- We expected humans to be doing most of the pipeline mapping, but we also envisioned software tools for pipeline analysis
  - So we wanted human and machine consumability
  - We biased slightly toward humans for 3 reasons
    - “Bias” like “syntactic sugar” and shortened OpenFlow keywords to reduce typing
    - Thought that would be best way to foster adoption
    - Some things we were not sure how to express “unambiguously”
    - Formalizing machine consumability would require a skills and time that we didn’t have

- We wanted > 1 common languages for the data structure
  - JSON, XML, etc

- The TTP spec uses JSON examples, but we intended for XML and possibly YANG, YAML, etc, to express TTPs also
Challenges and progress intermixed

GOOD NEWS!

• Before TTP spec even published, the market responded
  – Broadcom’s OF-DPA! in PowerPoint… and didn’t call it a TTP… oops
    • But JSON came fast and OFDPAv2 is a (huge) formal JSON TTP
    • OFDPA JSON is built by tools from complex Excel files… not directly human built
  – Early on (pre-OFDPAv2), OpenDaylight was enabled to import TTPs
    • Colin Dixon wrote YANG that was an effective schema for TTPs

  – But OFDPAv2 is not a use case model, it is a device model
    • Not mappable to other vendors
New Thinking

• We now have machine generation and consumption of TTPs
  – Humans like tools…
    • But tools like schemas and full openflow.h enum names
    • Also, schemas are fussy about their syntactic sugar…
  – Result: TTPv1.1, coming soon, will be firmly “schema friendly”

• And we now think in terms of both Use Case and Device models
TTP Tools...
“Enlightened” Approach

• A lot of time has passed
  – Controller world has, er, “simplified” in some ways
  – Controller players are now part of the WG
  – Broadcom’s device-oriented OFDPA concept changed the picture
  – Originally wanted Use Case models, but Device Models also valuable

• We need both kinds of pipeline models, and work them together
“Enlightened” Approach

• A lot of time has passed
  – Controller world has, er, “simplified” in some ways
  – Controller players are now part of the WG
  – Broadcom’s OFDPA concept changed the picture

• 2nd plan puts more work in the controller…that’s a better place
  – Rapid code changes are not hard, easier to iterate early versions
  – 3rd parties can experiment easily
  – Switch vendor can focus on switch-oriented code
Use Case TTPs very similar to Flow Objectives

Model Driven abstraction layers build the southbound mappings “automagically”

Device supports only one vendor-centric model

Mapping layer

Mapping layer
And finally: The Problem Statement

• SDN needs scalable, hardware independent platforms
• Interesting forwarding requires interesting pipelines
• The market wants a variety of pipeline models for use cases
• The available pipelines are diverse
• Mapping seems doable, but is an impediment

• Can we accelerate mapping?
Using ODL tool to compare models

- Not that hard to look at all the possible processing pathways through a pipeline model
  - Walk through the model, look at all the matches and actions
  - ODL utility is doing this now

- All pathways can be listed for Both the Use case, and the device
  - Use cases are simpler (devices try to support many uses)

- Once we have both lists, compare the two behaviors
  - Look for device pathways that offer what the use case requires
Mapping the pathways… scalably

• Mapping use case pathways onto devices is humans already do

• Work in ODL is oriented toward helping humans do the mapping
  – List all the pathways
  – Automate the search for matching pathways
  – Help a human figure out which device pathways work for a use case

• But why can’t this be done completely by machine?
Mapping layer

Device supports only one vendor-centric model

Use Case TTPs very similar to Flow Objectives

Model Driven abstraction layers build the southbound mappings “automagically”
Proposal: Header Space Analysis

- Taking a step back…
- We can do header space analysis on Use Case models
- And we can do the same on Device Models
- We should be able to, automatically, run through a variety of mappings and figure out which mappings are equivalent in HSA terms
Thank you