SDN Switch Inference and Control Plane Optimization

Daniel Tahara
daniel.tahara@yale.edu
Yale University

1 Background

Traditional network routing is decentralized, with entities generally using BGP to exchange routing information between autonomous systems and then some form of other routing protocol such as OSPF or RIP to exchange routing information within. Although this approach works well for public routing, it does not allow enough control for a network controller to implement various security and quality-of-service (QoS) policies within their own private network.

OpenFlow (OF) defines a protocol by which a centralized software controller can fill the routing tables of OF-enabled switches with forwarding rules defined by the network administrator. Specifically, these forwarding rules comprise sets of 12-tuples called flow entries, which support exact and wildcarding matching on the following fields: ingress port, ethernet src/dest/type, VLAN id/priority, IP src/dest/protocol/ToS bits, and TCP/UDP src/dest.

Wildcard rules introduce two fundamental design challenges in the OF API and switches. Because incoming packets can match multiple wildcarded rules, there needs to be some way for the OF switch to decide the correct forwarding action. The OF API reconciles this issue by requiring the controller to assign a priority value to each match condition. The switch then selects the matching rule with the highest priority. An important consequence of this is that hardware-based forwarding tables must have their entries sorted physically by priority in order to retain (software forwarding does not face this limitation since it can apply a logical filter to the rules after the fact).

The second design challenge occurs on the switch hardware level, since matching wildcarded rules on a hardware level requires customized hardware. OF switches typically rely on ternary content addressable memory (TCAM), which allows matching on 0, 1, or X (don’t care) for each bit. Although TCAM exists for normal forwarding tables, OF presents a significant challenge because the match conditions are about 250 bits wide\(^1\). Since TCAM takes up significantly more space than traditional CAM, this creates a severe limitation on the number of hardware entries a switch can support. As a result, most switches add a software-based flow table to handle more rules but at a significant performance cost.

2 Discussion

As a result of the two challenges mentioned above (among others), there is tremendous variation in hardware TCAM size, caching policies (which rules to include in the TCAM),

\(^1\)http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf
and general performance of each of the hardware and software layers (before we even introduce software-based, Open vSwitch (OVS) controllers). Although firewalls and other simple 'policy' controllers are unaffected by this diversity, controllers that cater to end applications cannot generalize about switch forwarding performance and thus cannot hope to accurately meet the QoS requirements of those applications.

However, if the controller can infer the critical properties of the switches as listed above, it can provide a better picture of the actual state of network performance. With that information, the controller could then either expose it to the programmer directly or make routing decisions on behalf of the user through some API. Rather than programming the switches, a programmer could truly ‘program the network,’ an explicit goal of SDN that, in our opinion, has yet to be met. Furthermore, if the controller can infer properties about the control plane, it can determine some optimal way of installing rules into the switches and minimize the overall impact to the network.

3 Implementation

A network-aware controller requires two major components, an inference engine and a scheduler/optimizer. The inference engine would generate various rules and traffic and measure installation and response times to infer the various control plane and dataplane properties mentioned above. A basic algorithm might do the following:

- Insert a rule
- Send traffic for that rule and measure the RTT
- If the RTT is higher than all previous times by a significant margin (> 3 standard deviations), we have found the cache size

This ‘edge detection’ algorithm may be correct in simple cases where the switch installs rules directly into TCAM and does not replace them if the TCAM overflows, but it might fail for more complex cache policies and general switch design. This algorithm would therefore vary based on the model for how a switch cache works.

The scheduler or optimizer would take the data generated during the inference phase and then compute network paths that satisfy given QoS requirements, as well as an installation plan for the generated rules. A sample routing algorithm might generate a subgraph of the network based on the links and nodes that meet the minimum bandwidth and latency requirements, and then use a greedy algorithm to generate an approximate shortest path. There are a number of tradeoffs with respect to how close the approximation is and the overall runtime of the algorithm, and these can be explored both formally and experimentally.

4 Deliverables

A lot of the groundwork has been done on this project in the preceding month, which culminated in a submission of an abstract the ONS 2014 [1] and a full paper to SIGCOMM 2014 [2]. In that work, I wrote the initial probing code and successfully inferred TCAM size and cache policy on a hardware switch. However, much work remains to be done, including improving and generalizing the inference code, and then extending it to include control plane properties. Additionally, after specifying an application API to the controller, I began to
develop algorithms for doing the rule generation based on QoS requirements (and extended routing algorithm) and scheduling for rule installation. My plan for the remainder of the semester is to formalize these two algorithms so that we can provide a complete application framework for others to potentially extend. Ideally, we would submit to another conference (pending decision from SIGCOMM) and open source our framework.

References
