Using P4 to Enable Scalable Intents in Software Defined Networks

Benjamin Lewis
Lancaster University
Lancaster, UK
b.lewis@lancaster.ac.uk

Lyndon Fawcett
Lancaster University
Lancaster, UK
l.fawcett1@lancaster.ac.uk

Dr. Matthew Broadbent
Lancaster University
Lancaster, UK
m.broadbent@lancaster.ac.uk

Prof. Nicholas Race
Lancaster University
Lancaster, UK
n.race@lancaster.ac.uk

Abstract—**When designing Software Defined Networks (SDNs), there is a risk that the additional abstractions available can result in reduced scalability and performance. One such abstraction, intents, are a way in which network administrators can express policies rather than having to define specific forwarding rules. This provides a benefit to administrators in allowing automatic network reconfiguration and fault tolerance. In this paper, we highlight the performance overheads associated with the intents framework from a popular SDN controller, ONOS. We propose a novel prototype that leverages source-based routing and programmable data planes using P4 in order to reduce the overheads of intent-based forwarding.**

Index Terms—**P4, SDN, OpenFlow, ONOS, intents**

I. Introduction

Intents are a mechanism network operators can use to provide policy-based routing, treating the network as an abstract series of direct connections between hosts, rather than as individual, interconnected forwarding devices. The role of an intent is in translating high-level policy into installable forwarding rules [1]. Due to intent processing complexity having an intrinsic relationship with the size of the network, intent computation time increases with network scale.

Current implementations of intents, such as those used in ONOS [2] take advantage of OpenFlow's control plane flexibility. However, this flexibility is bound by OpenFlow's fixed function data plane. The key element in our approach to scaling intents is the use of a programmable data plane provided by the P4 Language [3].

This paper studies the performance of the ONOS intent framework. It then goes on to propose a P4-based approach using Source Based Routing. Finally, we evaluate our approach, highlighting the performance benefits in intent installation time compared with OpenFlow-based solutions.

II. ONOS Intent Performance Analysis

Despite being an abstraction of the forwarding plane, intents are still 'compiled' into a set of forwarding rules, which may ultimately need to be installed onto every switch.

Initial tests were performed with ONOS to measure the time elapsed between sending an intent request and the associated installation (this was achieved using the ONOS metrics application, included within the core ONOS project). Tests were performed on an Intel Core i7 8700k (hex-core) CPU @ 3.7GHz running on an Ubuntu 16.04 host with ONOS v1.13 and the network emulation tool Mininet [4].

The tests were conducted to determine the impact on intent installation time when: a) adding additional controllers to an existing cluster and b) adding additional switches to a single controller.

Our analysis highlights a significant change in the intent installation time relative to the number of switches and controllers. Figure 1 shows that as the number of controllers increases, the installation time for a single intent also increases. This is due to the overhead of distributed state replication.

Figure 2 highlights the increase in intent installation time with the size of the network. This upwards trend in latency is due to a combination of increased path computation complexity with network size, and the increase in the number of forwarding devices that need to be communicated with.

III. P4 Source Routing Design

Source Based Routing is seen as an enabler to routing abstraction [5]. By using data available to the source host or switch, decisions about the desired path through the network can be made, and the packet will be forwarded according to rules appended to the packet header. Implementations have thus far used OpenFlow, coupled with specially crafted packet headers. In these cases, each packet contains the full route to the destination, with intermediate switches simply forwarding the packet according to the next hop (as contained in the packet header). However, this relies on the entire network state being known to the source.

This paper proposes the use of P4 [3] to overcome this limitation. For our experiments, we run the simple_switch_grpc of the P4 behavioural model. This enables P4Runtime support, a platform-independent API for programming P4 switches or their software based simulators. In this example, each switch has an independent controller, with each only needing to be aware of neighbouring switches and their available paths.

This approach separates the responsibility of maintaining information on links between source and destination switches, and instead relies on each switch's own controller to change its own forwarding rules to handle link failures or network congestion.

The packet processing pipeline (available as an implementation at [6]) operates as follows: The packet is received by

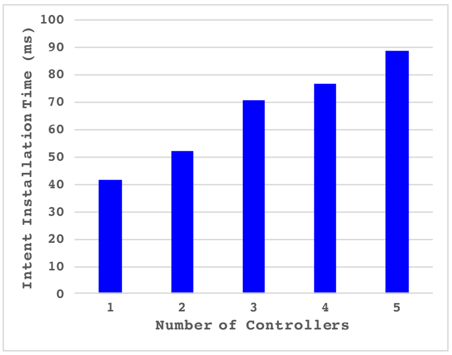

Fig. 1. ONOS intent installation time: multiple distributed controllers

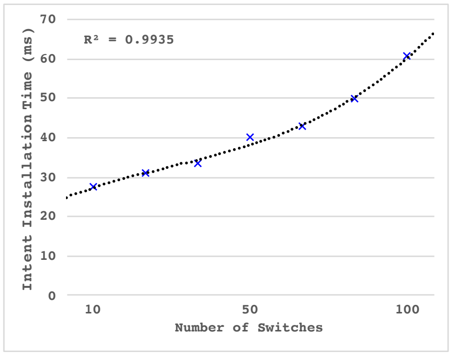

Fig. 2. ONOS intent installation time: Single controller

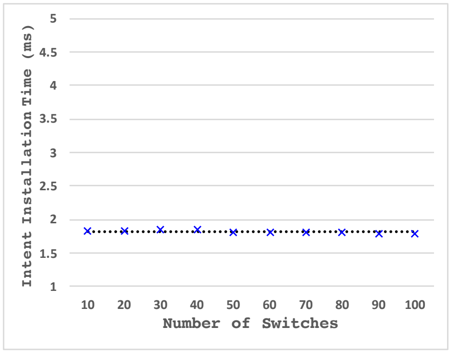

Fig. 3. P4 intent installation time

the switch and parsed to determine whether it is a normal Ethernet packet or has a specific forwarding header. In the case of the latter, the switch checks whether it is the destination specified, else it forwards the packet on to the next hop. This is determined by its own forwarding table.

In the case that the packet has a Layer 2 Ethernet header, it is checked against a match-action table to determine whether it should be treated as part of a host-to-host intent, at which point, a new header is added to the packet, with the destination set as the final destination switch for the intent. This is determined by the original rule installed by the controller.

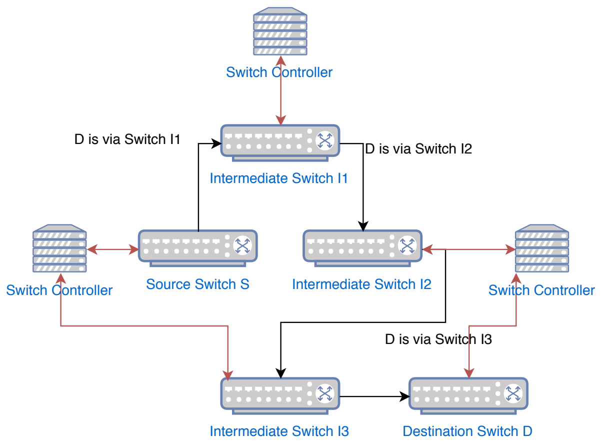

Fig. 4. P4 Source Based Routing overview

IV. P4 Intent Performance Analysis

We tested the performance by timing the installation of an intent to forward a packet from an initial switch (switch 0) to switch N (where switch N is the last switch in the chain of interconnected virtual switches). These tests were performed on the same machine described previously, using version 1.11.0-de8f08b0 of the behavioural model.

Performance figures collected show that the time taken to install a new rule does not significantly increase with the number of switches. This is because the only switch requiring a rule insertion is the source switch. In a worst case scenario, the destination switch may need a rule installed to return traffic, but it is possible to do this autonomously at the controller level. There is a small variation in the time to install a rule in the magnitude of ± 0.1ms and this can be attributed to system scheduling time.

A. Limitations of Source Routing

Whilst we provide a solution to the intent installation time, we do not yet offer an automated solution to link discovery or updates when link failures occur. These should only need to be handled directly on the links that have failed.

There is a small packet overhead added by the source routing header, but the header is only extended by 14 bytes, or approximately 1% of the MTU of an Ethernet frame.

V. Conclusion and Future Work

This paper has presented an alternative approach to policy-based intents in ONOS by using the P4 data plane programming language. Preliminary results show how this solution can scale without a significant performance overhead. Future work will include further analysis, incorporating network discovery, failover, and exploring our approach with tiered controllers to further improve scalability of SDN based solutions.

References

[1] M. Santuari. (2017) ONOS domain intent. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Domain+Intent
[2] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.
[3] P. Bosshart and Daly, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890
[4] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466
[5] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, "Source routed forwarding with software defined control, considerations and implications," in *Proceedings of the 2012 ACM conference on CoNEXT student workshop*. ACM, 2012, pp. 43–44.
[6] B. Lewis. (2018) Github - p4 source routing. [Online]. Available: https://github.com/BenRLewis/P4-Source-Routing