Turning the Tortoise to the Hare: An Alternative Perspective on Event Handling in SDN

Soheil Hassas Yeganeh and Yashar Ganjali
Department of Computer Science, University of Toronto
Toronto, Canada
soheil@cs.toronto.edu, yganjali@cs.toronto.edu

ABSTRACT

OpenFlow, the most popular software-defined networking proposal, decouples control from data path, and pushes control functionality to a software-based controller with a centralized view of the network. Such a design enables change and innovation by simplifying the implementation of control applications. A historically narrow interpretation of control and data planes, however, has led to unnecessary limitations on network applications that can be supported by OpenFlow. More specifically, applications handling frequent events are pushed outside the scope of SDN as: i) the controller cannot tolerate the high load; and ii) changing data path ASIC is an extremely costly and long process.

In this paper, we present our perspective on this challenge. We believe these are non-intrinsic limitations that can simply be viewed as design challenges in areas such as state abstraction, event propagation, parallelism, and placement of network applications. We show how a consolidated software platform can be used to address these challenges, and how this design leads to a cohesive framework capable of accommodating applications with frequent events.

Categories and Subject Descriptors

C.2 [**Computer-communication networks**]: Network Architecture and Design

Keywords

Software-Defined Networking; Distributed Control Platforms; OpenFlow

1. INTRODUCTION

Software-Defined Networking (SDN) is an attempt to overcome the rigidity of today's networks and to simplify change and innovation. It has been studied in the research community for some time [6, 1, 2], but started to gain attraction from industry after the introduction of OpenFlow [9]. OpenFlow was designed based on three premises: (*i*) control plane must be decoupled from the data path elements; (*ii*) control applications can be moved to a (logically) centralized controller; and (*iii*) the controller communicates with data path elements using well-defined APIs.

Separating control plane and data path elements allow the two to evolve independently. With a centralized controller, the design and implementation of control applications becomes very simple, as we do not need to worry about the complications of distributed systems. With a well-defined API for packet forwarding, data path functionality can be realized using a simple, low cost, and vendor-agnostic hardware substrate. Together, all these factors make innovation possible, and allow networks to evolve easily to match external requirements.

Data Path and Packet Forwarding. Traditionally, network routers and switches are composed of two main components: (*i*) a hardware-based forwarding fabric that forwards packets at line rate based on a set of rules, and (*ii*) a control software that installs and updates forwarding rules on the forwarding fabric. Different routers coordinate using messages exchanged between the control plane applications.

The initial design of OpenFlow adopts the definition and responsibilities of control plane and data path elements from their traditional counterparts in routers and switches. Data path elements are responsible for packet forwarding, which happens through a Forwarding Information Base (FIB). The controller updates the FIB and therefore manages how the network handles individual packets.

Where Can We Handle Frequent Events? OpenFlow's centralized control plane and its narrow interpretation of data path as a packet forwarding substrate impose strict limitations on network applications. Any functionality that requires frequent events (such as per packet processing) imposes a prohibitively large overhead on the control plane, and therefore cannot be implemented as a control application. On the other hand, changing the data path ASIC to add any new functionality for handling frequent events can be extremely costly, may take a very long time, and obviates most advantages of SDN.

In practice, these limitations have lead to the implicit assumption that any functionality requiring high frequency events must be pushed outside SDN.

Example 1: Elephant Flow Detection. Let us assume we are interested in identifying elephant flows in an OpenFlow network (in order to route them through a fat optical pipe, for instance). Implementing such a function in the control plane would require constant polling of network switches, which can put an extremely high load on the controller and control channels. DevoFlow [4] suggests

BigSystem 2014, June 23, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2609441.2609642.

changes to the forwarding ASIC in order to identify potential elephant flows on the switch and therefore to limit the overhead imposed on the control plane.

Example 2: Traffic Classification. Traffic classification is a complex network function with applications in QoS, security, and accounting. Most advanced traffic classification engines require Deep Packet Inspection (DPI) which clearly requires line rate packet processing, and therefore, cannot be handled in the OpenFlow control plane. On the other hand, the complex and dynamic nature of traffic classification techniques make it extremely costly, if not impossible, to push them to the ASIC as data path functionalities. As a result, it is commonly assumed that traffic classification must be handled outside SDN, for example as a middlebox.

In this paper, we portray a paradigm that enables us to handle frequent events within the scope of SDN. We associate the perceived limitations of SDN to a narrow interpretation of control plane and data path elements, and challenge the common belief that frequent events are either handled in data path, or they have to be rendered as "out-of-scope" for SDN.

In our view, rather than thinking about control vs. data path, we need to focus on development cycle, or simply software vs. hardware. We believe, new functions – even the ones dealing with frequent events – should emerge in software to reap the benefits of SDN including rapid innovation. This is, in essence, an established practice in the networking industry, shadowed by the control vs. data path design in oOpenFlow. Any efficiency drawbacks of such a design are non-intrinsic and must simply be viewed as design challenges in areas such as state abstraction, event propagation, parallelism, and placement of network applications. Here, we sketch the design landscape for a consolidated software a platform that can address these challenges.

2. PROBLEM STATEMENT

While SDN promises to enable innovation in networks, it is not clear where and how a non-forwarding networking function can handle frequent events. In essence, thinking of SDN within the confines of the original OpenFlow (*i.e.*, limiting networks to connectivity and forwarding) can be quite limiting when adopting SDN in new domains. In this section, we first elaborate on this argument and then present an objective methodology on how to introduce and evolve new networking functions in SDN that can handle frequent events at scale.

In Control Plane. The easiest way to implement any new function in SDN is to develop the function as a control application running on the controller. With a centralized view of the networks, developing control applications become profoundly simpler than developing a similar logic as a colony of fully distributed agents running on routers. This simplicity makes this option *the* most natural path to enable innovation and change in SDN.

Unfortunately, this simplicity can come at a high cost: providing a centralized view can be quite challenging large-scale networks, especially if updates are frequent. No matter how powerful the controller is, processing frequent events in a centralized manner can create scaling bottlenecks. This can be a significant barrier for introducing any new function in the control plane, and is the reason why some proposals (such as DIFANE [11] and DevoFlow [4]) introduce new frequent event processing functionalities into the forwarding ASICs.

In Forwarding Silicone. Processing frequent events using hardware primitives in forwarding silicone can result in considerably better efficiency and lower costs at scale compared to a centralized controller. This method, however, comes with its own pitfalls: (*i*) Designing a special purpose hardware for a new function imposes a rigid structure; hence, that function cannot evolve frequently. If we need to update the design design (say to fix an error, or to improve performance) we need to wait for a considerably long time for the next development cycle. (*ii*) Design, development, and manufacturing special purpose hardware is quite costly (orders of magnitude more expensive than software), which increases the risk for already risk-averse ASIC manufacturers.

Because of these pitfalls, silicone is not the best way for introducing new network functions, where chances of change is not negligible. In essence, it would make sense to use silicone for implementing a specific function only when (*i*) the function is mature enough (*i.e.*, applied and tested in real-world for quite some time), and (*ii*) we have hit a performance barrier on commodity hardware. This explains OpenFlow's focus on forwarding as the main data path functionality.

Outside the Scope of SDN. Looking at the drawbacks of the two options of silicone and centralized controller, one can argue that a network function should be introduced outside the scope of SDN if it is both complex (*i.e.*, difficult to implement inside silicone) and requires processing frequent events (*e.g.*, per packet processing).

Based on this orgument, some proposals introduce packet processing in software modeled as black-boxes sitting either at the core or at the edge of the network. Other proposals focus on processing resources available inside switches such as co-processors and FPGAs. The common trait among these proposals is that they all rely on SDN solely for steering packets through black-boxes and packet processing units.

Furthermore, we can also handle frequent events in end-hosts if we have access to them and the functionality can be implemented without significant coordination overhead. This is, for example, the path taken by Open vSwitch [10]. This is a viable option for networking environments such as a data center, but it would not be applicable in other environments such a carrier or an enterprise network.

To compare these alternative solutions, let us see how they can be used to implement a simple DPI-based Traffic Classification.

Traffic Classification on a Controller. To classify traffic in a control application, we need to process the payload of the first few packets of each flow in the control plane. Note that by the first few packets, we mean packets after the transport protocol handshake (*e.g.*, after TCP SYN-ACK). To do so, we need to install flow-entries in switches only if a flow is already classified. Even if we assume that a centralized controller has enough capacity to process packets at scale, sending the first packets of all flows to a centralized controller results in a huge load on the control channels and adds a considerable latency to all network flows.

Traffic Classification in Silicone. Traffic classification is a complicated task and implementing it purely in hardware would be quite challenging for many reasons. First of all, to

match a flow against a set of regular expressions, we need to preserve a large state for each flow since the matching part of the payload can be split between two packets. Even if we relax this constraint and match regular expressions only on single packets (*i.e.*, a stateless classification), we need to limit the size of the payload and also limit the number of regular expressions. Otherwise, the cost of manufacturing a hardware that can match large packets against an arbitrarily number of regular expressions can be sky rocketing. Due to these limitations, in practice, all ASICs with DPI capabilities are stateless and can only inspect the first few bytes of the payload (*e.g.*, roughly the first 100 bytes) against a few regular expressions (*e.g.*, tens of simplified regular expressions).

Traffic Classification outside SDN. The other alternative is to implement traffic classification in software running on either commodity hardware (*e.g.*, x86) or programmable hardware (*e.g.*, FPGA) in the data path. Such a software is fed with a set of protocol signatures to match against flows, and a set of actions to apply on the packets once the flow is classified. Flows that are required to be classified will be steered through these classifiers by a SDN control application. Moreover, the output of the classifier is also inspected to take further forwarding actions.

Although prevalent in traditional networks, this approach has profound pitfalls. Forwarding decisions are subpar and are coupled with the physical placement of middleboxes. That is, the physical topology of these classifiers dictates how the flows should be steered by SDN. Moreover, it is difficult to coordinate different classifiers in a large-scale network due to the lack of control over middleboxes. In essence, opting for this solution means losing the ability to refactor and optimize network services, and difficulties in troubleshooting and management, since we have poor visibility over a middlebox.

Traffic Classification in the End-Hosts. The best option for implementing traffic classification might be in the end-host, assuming it can keep up with the required processing at line rate. Unfortunately, this option does not exist when we do not have access to end-hosts in the network.

In a truly software-defined network, these alternatives (Figure 1) should be limited to a straightforward selection between silicone and software. On a retrospective, these solutions are still around just because of the historic evolution and limited scope of existing SDN proposals. Based on this view, we propose a methodology that moves in that direction.

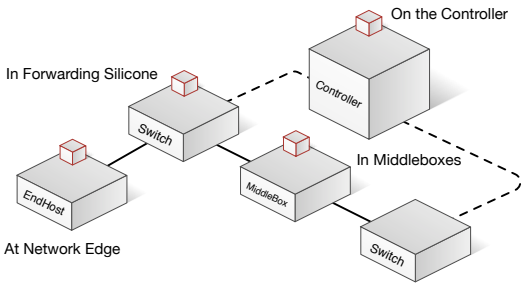

Figure 1: There are three major alternatives to implement new functions in SDN: (*i*) On the controller as a control application; (*ii*) In forwarding silicone inside switches; (*iii*) Outside SDN (*e.g.*, On middle boxes or on end-hosts).

3. AN ALTERNATIVE PERSPECTIVE

We believe one needs to clearly distinguish between intrinsic limitations of a SDN proposal as opposed to constraints resulting from its historic evolution. In our view, there is no inherent reason a controller should not be able to handle frequent events. This assumption comes from the way OpenFlow was historically presented: we need to have a physically-isolated, centralized controller, or at least a centralized view, and it is very challenging to handle frequent events in any system that provides a centralized view. Recent proposals, like Kandoo [7], show how we can handle frequent events in control plane.

Clarifying the difference between intrinsic limitations and challenges helps us gain a better understanding of the solution space, and not to needlessly eliminate viable solutions. For example, implementing new functions in silicone is by nature a costly and lengthy process. This is a limitation which cannot be overcome. In contrast, improving the performance of a software-based packet processing solution is a challenge. We might be able to handle this by code optimization, or by employing parallelism (*i.e.*, throwing more resources).

When it comes to frequent events (or any function beyond forwarding), the paradigm of separating control from data path should be shifted to separation of software from data path. In this model, frequent events can be handled using software (similar to control applications in OpenFlow). Obviously, for handling frequent events, we need an apt software platform with smart placement strategies and distribution mechanisms to gain scale and performance. Such a platform should be elastic enough to take advantage of any additional resource to optimize the performance of the system.

Of course, even with the best resource provisioning and code optimization, we might not be able to match silicone's behavior in some cases. However, we believe accepting this performance bound as a limitation is a more natural way than assuming the function can only be implemented in ASIC, which most likely will stop any further development.

In this model, hardware solutions will naturally emerge during the evolution stages, as mature functions are pushed to hardware with the objective of accelerate common parts of new networking functions. Using these hardware primitives one can accelerate parts of a network function without taking the risks of switching to hardware at an early stage.

These design challenges are mainly about scale and resilience in the software platform that hosts network functions (*a.k.a.* Control Plane in OpenFlow's terminology). As we shortly discuss, a software platform that provides arbitrary network functions at scale with the best use of the available computing resources would render all these so-called limitations irrelevant.

A Consolidated Software Platform. To enable our vision on SDN functions, we need a distributed software platform for network programming that have three main properties:

1. **Optimized Placement**: A key for a scalable SDN is to place network functions in the closest vicinity of event sources. For instance, a SDN platform can easily scale as long as it performs packet processing as close as possible to the edge. At the same time, a forwarding function that requires a centralized view of the network

should be placed in a way to minimize the tail latency to switches.

2. **High Resource Utilization**: Today's networks are equipped with a variety of computing resources. On the end-hosts, we have general purpose processors and GPUs, at the core we have general purpose processors and programmable hardware in switches. It would be quite beneficial when a consolidated software platform can offload networking functions on these computing power.

3. **Easy to Use**: The software platform must be easy to use, otherwise, if it makes it subtle and complicated, network programmers will never switch to that software platform. That is, unless we provide simplifying programming abstractions, network programmers would prefer to develop their functions in isolated silos rather than implementing it as a SDN function. Note that, in contrary to the popular belief, centralization is not the only way to provide an easy-to-use platform. In other fields, we have seen proposals such as MapReduce [5], Spanner [3], and Spark [12] that provide simplicity at scale by hiding generic boilerplates and complications.

Example: DPI. Using such a platform, one can implement DPI simply as a function that processes incoming packets regardless of the actual placement of the function in the wild. When deployed, the platform automatically pushes the functions as close as possible to the points where packets enter the network (*e.g.*, end-hosts if we are using soft switches, or choke points in a WAN) to scale. Moreover, since the platform is aware of resources, it would try to place the function on a NetFPGA over an x86 processor if possible for accelerated event handling.

Existing Proposals. There are a few recent software-based proposals to handle frequent events in SDN. For instance, Kandoo [7] categorizes control application into local (*i.e.*, applications that require the state of a single switch) and non-local applications. Local applications do not share state and as such they have implicit parallelism. Exploiting that property, Kandoo offloads local application to local processing resources (*e.g.*, end-hosts and co-processors on switches) and, as long as frequent events are processed in local applications, Kandoo simply scales. The problem with Kandoo is that it is not generic enough to help functions that require a state beyond a single switch.

Nicira NVP [8] is SDN-based network virtualization platform that provides packet processing in software at specialized gateway nodes. These gateways are connected to virtual ports (connecting hypervisors) using an overlay network. As such, NVP utilizes processing resources available in a data center to provide generic networking services such as firewalls and gateways. Although NVP's design is limited to virtualization environments (where everything goes through hypervisors) and does not utilize hardware resources available on physical switches, it demonstrates how one can introduce radically innovative functions in software without waiting for the hardware to catch up.

These recent proposals, although being very good starting points, do *not* cover all the requirements we envision of a complete SDN software platform. We believe developing this platform is still a significant open design challenge in SDN, that deserves extensive discussion and community's attention.

Coupling vs Collocation. Our methodology and our arguments relies on offloading functionalities close to switches to gain scale and performance. This might look like as coupling data and control planes back together again. On the contrary, we note that collocating software with forwarding fabric does not contradict decoupling. We indeed keep the control plane decoupled but, if the environment allows, we colocate them to reduce the overheads of having physically separate control and data planes. That is, the network applications can still function without being collocated with the data plane. This approach sits at the sweet spot in between traditional networks and OpenFlow: A decoupled control plane with minimal overheads.

4. CONCLUDING REMARKS

In this paper, we analyzed the problem of introducing a new networking function in SDN. We believe some of the design limitations are not inherent in SDN, and are only stemmed from the historical evolution of OpenFlow. Such limitations can be easily overcome using an aptly design software platform that can accommodate network functions ranging from packet processing to network control. Such a platform would be able scale by placing network functions on best available resources based on their characteristics.

5. REFERENCES

[1] M. Casado et al. Sane: A protection architecture for enterprise networks. In *Proceedings of USENIX-SS'06*, 2006.

[2] M. Casado et al. Ethane: Taking control of the enterprise. In *Proceedings of SIGCOMM'07*, pages 1–12, 2007.

[3] J. Corbett et al. Spanner: Google's globally-distributed database. In *Proceedings of OSDI'12*, pages 251–264, Berkeley, CA, USA, 2012.

[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *Proceedings of SIGCOMM'11*, pages 254–265, 2011.

[5] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of OSDI'04*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[6] A. Greenberg et al. A clean slate 4d approach to network control and management. *SIGCOMM CCR*, 35(5):41–54, Oct. 2005.

[7] S. Hassas Yeganeh and Y. Ganjali. Kandoo: A framework for efficient and scalable offloading of control applications. In *Proceedings of HotSDN'12*, pages 19–24, 2012.

[8] T. Koponen et al. Network virtualization in multi-tenant datacenters. In *Proceedings of NSDI'14*, pages 203–216, 2014.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74, Mar. 2008.

[10] B. Pfaff et al. Extending networking into the virtualization layer. In *HotNets*, 2009.

[11] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable flow-based networking with difane. In *Proceedings of SIGCOMM'10*, pages 351–362, 2010.

[12] M. Zaharia et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of NSDI'12*, 2012.