On Scalability of Software-Defined Networking

Soheil Hassas Yeganeh, Amin Tootoonchian, and Yashar Ganjali, University of Toronto

ABSTRACT

In this article, we deconstruct scalability concerns in software-defined networking and argue that they are not unique to SDN. We explore the often voiced concerns in different settings, discuss scalability trade-offs in the SDN design space, and present some recent research on SDN scalability. Moreover, we enumerate overlooked yet important opportunities and challenges in scalability beyond the commonly used performance metrics.

INTRODUCTION

Moving the control function out of data plane elements is the common denominator among software-defined networking (SDN) proposals in the research community. This decoupling enables both planes to evolve independently, and brings about numerous advantages such as high flexibility, being vendor-agnostic, programmability, and the possibility of realizing a centralized network view. Despite all these advantages, there have always been concerns about performance and scalability since its inception.

The common perception that control in SDN is centralized leads to concerns about SDN scalability and resiliency. After all, regardless of the controller capability, a central controller does not scale as the network grows (increase the number of switches, flows, bandwidth, etc.) and will fail to handle all the incoming requests while providing the same service guarantees. Additionally, early benchmarks on NOX (the first SDN controller), which showed it could only handle 30,000 flow initiations per second [1] while maintaining a sub-10 ms flow install time, intensified such concerns. Moreover, since most early SDN proposals were flow-based, additional flow initiation delay became a concern.

We argue that there is no inherent bottleneck to SDN scalability; that is, these concerns stem from implicit and extrinsic assumptions. For instance, the low flow setup throughput in NOX is shown to be an implementation artifact; and the control plane being centralized was simply due to the historical evolution of SDN. Without such assumptions, similar to any distributed system, one can design a scalable SDN control plane [2].

We believe there are legitimate concerns for SDN scalability. Interestingly, we argue that these scalability limitations are not restricted to SDN; traditional control protocol design also faces the same challenges. While this does not address these scalability issues, it shows that we do not need to worry about scalability in SDN more than we do for traditional networks.

ROOTS OF SCALABILITY CONCERNS IN SDN

What fundamentally differentiates SDN from traditional data networks is the separation of control from the forwarding plane. This decoupling leads to interesting properties. Most important, as exemplified in Fig. 1, data and control plane components can evolve independently, as long as we define a standard application programming interface (API) between the two. Forwarding elements are responsible for switching and are usually built from highly specialized application-specific integrated circuits (ASICs), which evolve significantly slower than control plane components written in software. Also, the *possibility* of creating a centralized view of the network creates tremendous potential opportunities for simplifying the control applications, and therefore accelerating change and innovation in the control plane.

This decoupling, however, has its own pitfalls too. First, defining a standard API between the two planes is absolutely nontrivial. Technically, this API should be able to handle the needs of various architectures, and should be able to facilitate the independent evolution of both planes. Also, all or a majority of switch vendors should adopt the same standard API for it to be useful; otherwise, networks will be tied to specific vendors, which might lead to proprietary layers, preventing rapid change and innovation in networks. Second, decoupling data and control planes brings about scalability concerns. Moving traditionally local control functionalities to a remote controller can potentially result in new bottlenecks. It can also lead to signaling overheads that can be significant depending on the type of network and associated applications.

In what follows, we first discuss SDN controller scalability, outlining why it has been a concern, and recent works in this domain. Then we review some other often voiced SDN scalability concerns including the flow setup overhead and resilience to failures. We argue that, even

0163-6804/13/$25.00 © 2013 IEEE

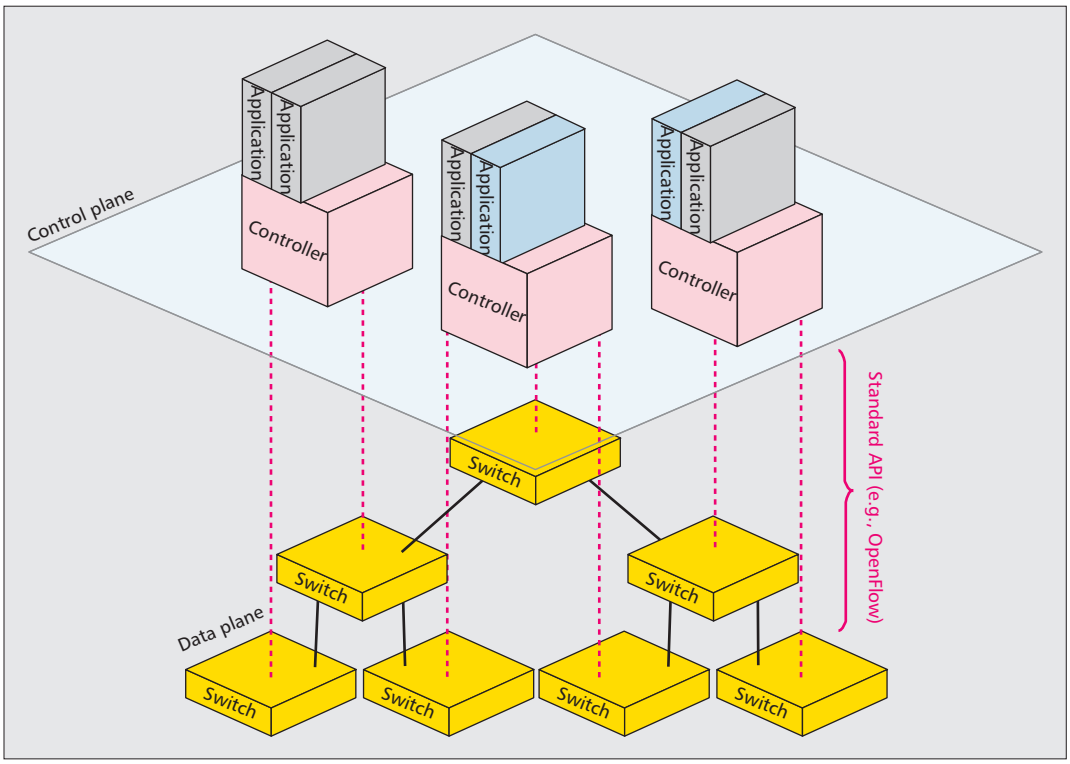

Figure 1. *Decoupled data and control planes in SDN can have different topologies, use different technologies, and, more important, evolve independently.*

What fundamentally differentiates SDN from traditional data networks is the separation of control from forwarding plane. This decoupling leads to interesting properties. Most importantly, data and control plane components can evolve independently, as long as we define a standard API between the two.

though these concerns are not specific to SDN, they can be alleviated with an alternative design (some of which are now commonplace).

Controller Scalability

One possible SDN design is to push all the control functionality to a centralized controller. Empowered with a complete network-wide view, developing control applications and enforcing policies become much easier in this setting. Nevertheless, controllers can potentially become the bottleneck in the network operation. As the size of a network grows, more events and requests are sent to the controller, and at some point, the controller cannot handle all the incoming requests. Early benchmarks of an SDN controller (NOX) showed that it can handle 30k requests/s [1]. Even though this may be sufficient for a sizeable enterprise network, it could be a major problem for data-center-like environments with high flow initiation rates [3].

One way to alleviate this concern is to level parallelism in multicore systems and improve IO performance. Tootoonchian *et al.* [4] showed that simple modifications to the NOX controller boosts its performance by an order of magnitude on a single core. It means that a single controller can support a far larger network, given sufficient controller channel bandwidth with acceptable latency. We can also reduce the number of requests forwarded to the controller. DIFANE [5] proactively pushes all state to the data path (addressing the scarcity of data path memory in a scalable manner). In DevoFlow [6], with support from an ASIC, short-lived flows are handled in the data path, and only larger flows are forwarded to the controller, effectively reducing the load on the controller and improving scalability. Therefore, we can say that DevoFlow trades fine-grained flow-level visibility in the control plane with scalability, which can be reasonable depending on the setting and constraints of the underlying network.

Alternatively, one can distribute the state and/or computation of the control functionality over multiple controllers. Having a centralized controller is by no means an intrinsic characteristic of SDN. All we need is a unified network-wide view to reap the benefits of SDN. We note that, similar to any distributed system, providing a strictly consistent centralized view can hinder response time and throughput as the network scales. As stated in [7], it is not always feasible to achieve strong consistency while maintaining availability and partition tolerance. Therefore, selecting an apt consistency level is an important design trade-off in SDN. To preserve scalability, one should design control applications with the weakest possible consistency level.

There are solutions where we can physically distribute the control plane elements, yet maintain the network-wide view. Onix [2], for example, is a distributed control platform that facilitates implementation of distributed control planes. It provides control applications with a set of general APIs to facilitate access to network state (NIB), which is distributed over Onix instances. HyperFlow [8], on the other hand, synchronizes network state among multiple controller instances, giving the control applications (running on every controller instance) an illusion of control over the whole network. This keeps the simplicity of developing the control plane on a central controller while alleviating a class of

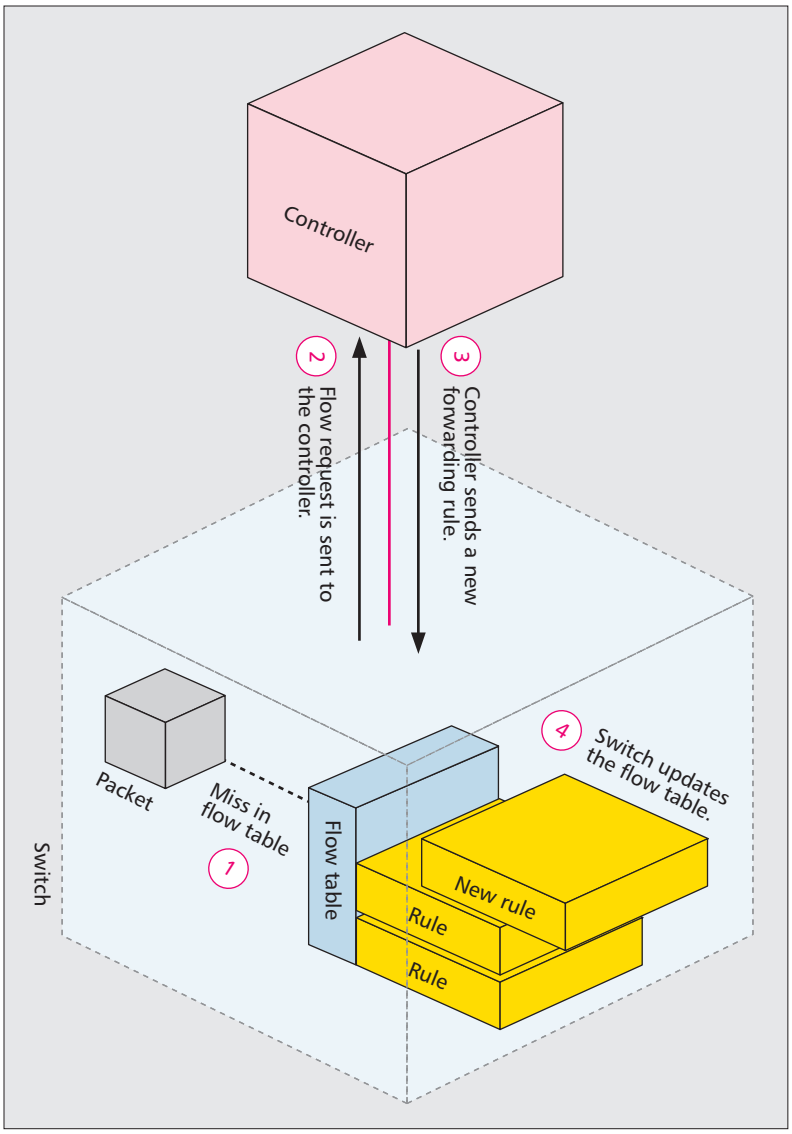

Figure 2. *The four steps in the flow setup process.*

scalability issues associated with a centralized controller, albeit for a more restricted set of control applications satisfying certain properties.

Kandoo [9] takes a different approach to distributing the control plane. It defines a scope of operations to enable applications with different requirements to coexist: locally scoped applications (i.e., applications that can operate using the local state of a switch) are deployed close to the data path in order to process frequent requests and shield other parts of the control plane from the load. A root controller, on the other hand, takes charge of applications that require network-wide state, and also acts as a mediator for any coordination required between local controllers.

An interesting observation is that control plane scalability challenges in SDN (e.g., convergence and consistency requirements) are not inherently different than those faced in traditional network design. SDN, by itself, is neither likely to eliminate the control plane design complexity or make it more or less scalable.[1] SDN, however:

- Allows us to rethink the constraints traditionally imposed on control protocol designs (e.g., a fixed distribution model) and decide on our own trade-offs in the design space
- Encourages us to apply common software and distributed systems development practices to simplify development, verification, and debugging

Unlike traditional networks, in SDN, we do not need to address basic but challenging issues like topology discovery, state distribution, and resiliency over and over again. As demonstrated in Onix, control applications can rely on the control platform to provide these common functions; functions such as maintaining a cohesive view of the network in a distributed and scalable fashion. In fact, it is significantly easier to develop applications for such cohesive distributed control platforms than a swarm of autonomous applications running on heterogeneous forwarding elements.

Other SDN Scalability Concerns

Increased load on the controller is only one of the voiced concerns about SDN scalability. Here, we briefly explain other causes of concern, along with potential solutions.

Flow Initiation Overhead — Ethane [10], an early SDN security system, puts a controller in charge of installing forwarding state on switches on a per-flow basis. Even though this *reactive* form of flow handling introduces a great degree of flexibility (e.g., easy fine-grained high-level network-wide policy enforcement in the case of Ethane), it introduces a flow setup delay and, depending on implementation, may limit scalability. Early designs, such as Ethane and NOX, lead to the widespread assumption that all SDN systems are reactive. In reality, however, proactive designs — in which forwarding entries are set up before the initiation of actual flows — are perfectly acceptable in SDN, and can avoid the flow setup delay penalty altogether.

Let us review the flow setup process to explain the bottlenecks and show how a good design can avoid them. As illustrated in Fig. 2, the flow setup process has four steps:

- A packet arrives at the switch that does not match any existing entry in the flow table.
- The switch generates a new flow request to the controller.
- The controller responds with a new forwarding rule.
- The switch updates its flow table.

The performance in the first three steps and partially the last depends on the switch capabilities and resources (management CPU, memory, etc.) and the performance of its software stack. The delay in the third step is determined by the controller's resources along with the control program's performance. Finally, the switch's FIB update time contributes to the delay in completing the flow setup process.

Assuming controllers are placed in close proximity of switches, the controller-switch communication delay is negligible. On the controller side, even on a commodity machine with a single CPU core, state-of-the-art controllers are well capable of responding to flow setup requests

[1] *After all, one can replicate a traditional network design with SDN by collocating equal numbers of forwarding and control elements. Even though this obviates the benefits of SDN, it is technically possible.*

within a millisecond when the flow initiation requests are on the order of several hundred thousand per second.

While Open vSwitch — an OpenFlow-enabled software switch — is capable of installing tens of thousands of flows per second with sub-millisecond latency, hardware switches only support a few thousand installations per second with a sub-10 ms latency at best. This poor performance is typically attributed to lack of resources on switches (weak management CPUs), poor support for high-frequency communication between the switching chipset and the management CPU, and non-optimal software implementations. We expect these issues to be resolved in a few years as more specialized hardware is built. It is foreseeable that the FIB update time will become the main factor in the switch-side flow setup latency.

While we argue that controllers and, in the near future, switches would be able to sustain sufficient throughput with negligible latency for reactive flow setup, in the end the control logic determines the scalability of a reactive design. A control program installing an end-to-end path on a per-flow basis does not scale, because the per-switch memory is fixed but the number of forwarding entries in the data path grows with the number of active flows in the network. However, the control program may install aggregate rules matching a large number of micro-flows (thereby facing the same scalability challenges as a proactive design), or proactively install rules in the network core to provide end-to-end connectivity and identify quality of service (QoS) classes, while classifying and reactively labeling flows at the edge. A viable solution to the scalability challenges of the proactive designs in the former class due to data path memory scarcity is proposed in DIFANE [5]; while the scalability of the latter class follows from the observation that the fanout of an edge switch and thus the number of flows initiated there is bounded (just add edge controllers as the network grows in size).

Resiliency to Failures — Resiliency to failures and convergence time after a failure have always been a key concern in network performance. SDN is no exception, and, with the early systems setting an example of designs with a single central control, resiliency to failures has been a major concern. A state-synchronized slave controller would be sufficient to recover from controller failures, but a network partition would leave half of the network brainless. In a multi-controller network, with an appropriate controller discovery mechanisms, switches can always discover a controller if one exists within their partition. Therefore, given a scalable discovery mechanism, controller failures do not pose a challenge to SDN scalability.

Let us decompose the process of repairing a broken link or switch to see how it is different from the traditional networks. As shown in Fig. 3, convergence in response to a link failure has five steps. The switch *detects* a change. Then the switch *notifies* the controller. Upon notification, the control program *computes* the repair actions and *pushes* updates to the affected data path elements, which, in turn, *update* their forwarding tables.[2] In traditional networks, link failure notifications are flooded across the network, whereas with SDN, this information is sent directly to a controller. Therefore, the information propagation delay in SDN is no worse than in traditional networks. Also, as an advantage for SDN, the computation is carried out on more capable controller machines as opposed to weak management CPUs of all switches, regardless of whether they are affected by the failure or not.

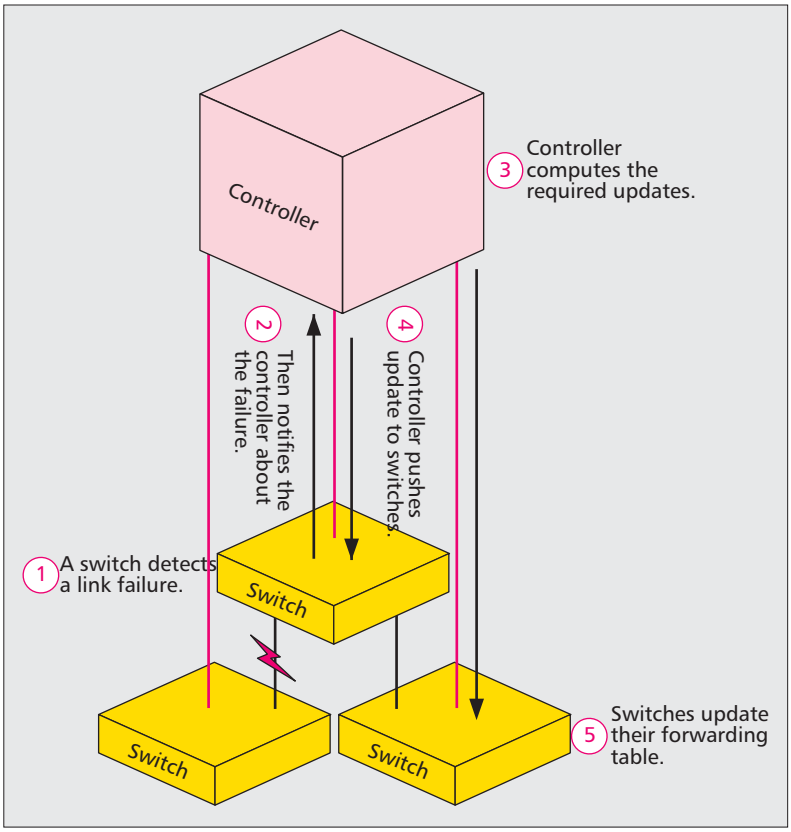

Figure 3. *The five steps when converging on a link failure.*

Note that the above argument was built on the implicit assumption that the failed switch or link does not affect the switch-controller communication channel. The control network itself needs to be repaired first if a failed link or switch was part of it. In that case, if the control network — built with traditional network gear — is running an IGP, the IGP needs to converge first before switches can communicate with the controller to repair the data network. In this corner case, therefore, convergence may be slower than in traditional networks. If this proves to be a problem, the network operator should deploy an *out-of-band* control network to alleviate this issue.

Overall, the failure recovery process in SDN is no worse than in traditional networks. Consequently, similar scalability concerns exist, and the same techniques used to minimize downtime in traditional networks are applicable to SDN. For instance, SDN design can and should also leverage local fast failover mechanisms available in switches to transiently forward traffic toward preprogrammed backup paths while a failure is being addressed. We stress that, as demonstrated in Onix [2], the control platform provides the essential failover and recovery mechanisms that control applications can reuse and rely upon.

[2] *For switch failures, the process is very similar with the exception that the controller itself detects the failure.*

A typical data center network has tens of thousands of switching elements and can grow at a fast pace. The sheer number of control events generated in any network with that scale is enough to overload any centralized controller. One way to tackle that problem is to proactively install rules on switches.

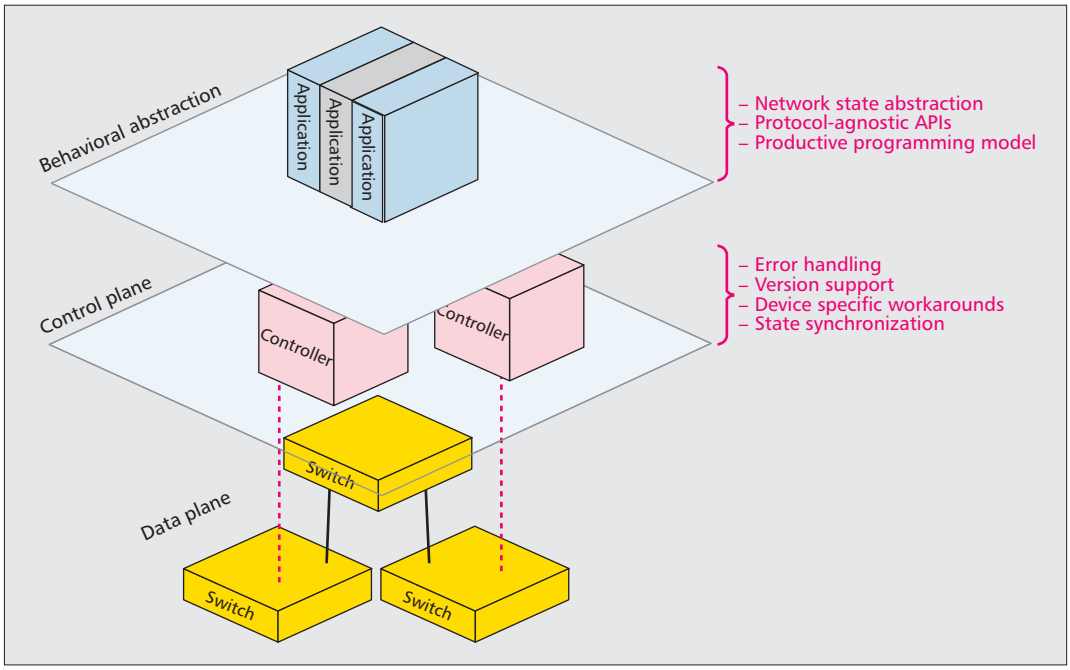

Figure 4. *SDN requires more productive management APIs that does not leak the complexities of south bound standard APIs to control applications and other management systems.*

Scalability in Different Network Settings

So far we have explored the most important scalability concerns in SDN by identifying various metrics that can potentially be affected as the network grows. In this section, we take a different approach by explaining scalability patterns and pitfalls in different types of networks. We have chosen data center and service provider networks as examples since they present diverse settings and different requirements. We do not aim at providing an exhaustive list of networks here. The goal is to identify patterns that can be generalized and used as references in understanding scalability issues in other networks.

Data Centers — A typical data center network has tens of thousands of switching elements and can grow at a fast pace. The sheer number of control events generated in any network at that scale is enough to overload any centralized controller. One way to tackle that problem is to proactively install rules on switches, effectively eliminating most control requests before they enter the control plane. The obvious cost here is, of course, loss of precision and reactivity in the controller.

When an application requires accurate flow statistics and/or reactivity, one can deploy the application close to the switches. For instance, frequent events can be delegated to processes running on end hosts as long as access to global state is minimized. Given the availability of processing resources throughout data center networks, solutions such as Kandoo [9] can be used to reach arbitrary scalability levels. Distributed controllers (e.g., HyperFlow or Onix) can also be reasonable solutions in data center networks. Given the low latency in such networks, synchronization of state and flow setup latencies would be minimal and acceptable for most applications.

Service Provider Networks — Typically, service provider networks do not have as many switches/routers as data center networks; however, nodes in such networks are usually geographically distributed. The large diameter of these networks exacerbates controller scalability concerns, flow setup and state convergence latencies, and consistency requirements. We might be able to take advantage of the physical distribution of the network to partition it into separate regions; each partition can be controlled by an independent controller, and these controllers can exchange only the required state changing events, effectively hiding most events from external controllers. Given the intrinsic delay in such networks, all the control applications should be latency tolerant and have weak consistency requirements.

In addition to high latencies, service provider networks usually have larger numbers of flows than other networks. As a result, data path resource limits are also of concern here. Aggregation of flows is the simple solution, which comes at the cost of granularity in control. We note that these concerns are also present in traditional networks, and are not unique to SDN.

Opportunities and Challenges

Traditionally, scalability of a network is studied based on performance metrics, that is, how a certain performance measure changes as we scale the network along a given dimension. In practice, there are other orthogonal aspects that profoundly affect how a system can accommodate growth. For instance, manageability (how convenient it is to manage a network when networking elements are added, removed, or modified at large scales) and functional scalability

(how much effort it takes to add a new functionality to the network) are as important as performance for network scalability, and should not be overlooked. Preliminary results on behavioral and programming abstractions, testing, and verification, as well as extensibility of SDN show this is an area in which we believe SDN presents a significant opportunity. Clearly, we still have major challenges in each of these aspects before we can reach the full potential of SDN.

Behavioral and Programming Abstractions — Almost all SDN controllers provide interfaces that expose the details embedded in the API between the data and control planes. That level of detail is essential for implementing a controller, but not generally required for managing a network. Network operators and programmers instead require an abstraction through which they can specify the required behavior from the network with minimal effort. To that end, as shown in Fig. 4, most details and complexities should be handled by the controllers and hidden from management interfaces. Such an interface will accommodate network growth and ease management at scale. This is a major challenge for the future of SDN, and (initial) solutions such as Frenetic [11] are only a glimpse of what is to come.

Testing and Verification — Without the right tools in place, troubleshooting becomes a major problem for networks as they grow. This is a major challenge for all types of networks, including SDN. The good news is that SDN provides building blocks for network testing that are not readily available in traditional networks. Moreover, given a control application, one can model its behavior and analytically verify it for an arbitrary network. Having said that, considering the complexity of networks, it is not trivial to realize a tool that can test and verify at scale. HeaderSpace Analysis [12, 13] is an example of efforts moving in that direction. Given the gap between the tools and the requirements, this topic remains a challenging problem.

Extensibility — Current SDN implementations take a pragmatic approach in supporting only well-known popular protocols that are readily supported by most equipment and chip vendors. As SDN scales, new protocols, or new versions of existing protocols, will get in the way. To support those new protocols, one needs to modify existing APIs. A solution for this problem would be a more extensible and expressive API between the control and data planes, which is quite challenging. The challenge is to design expressive constructs that can be implemented in ASICs, and support unforeseen future protocols and requirements. Even though this is not a direct scalability problem, it can have major impact on the growth and extensibility of SDN in practice.

Conclusion

Scalability has been a major concern since the introduction of SDN. The current body of research on SDN scalability shows that:

- These concerns are neither caused by nor fundamentally unique to SDN.
- Most of these issues can be addressed without losing the benefits of SDN.

Current warehouse scale SDN deployments support this argument.

What is usually overlooked in this domain is the impact of SDN on other limiting factors for the growth of networks such as network programming and management complexity. Software-defined networking adds a level of flexibility that can accommodate network programming and management at scale. Traditional networks have historically failed in this area. Recent attempts in this direction are very promising, even though many challenges remain for the future.

Software-defined networking adds a level of flexibility that can accommodate network programming and management at scale. Traditional networks have historically failed in this area. Recent attempts in this direction are very promising, even though many challenges remain for the future.

References

[1] A. Tavakkoli *et al.*, "Applying NOX to the Datacenter," *Proc. ACM HotNets-VIII Wksp.*, 2009.
[2] T. Koponen *et al.*, "Onix: A Distributed Control Platform for Large-Scale Production Networks," *Proc. 9th USENIX OSDI Conf.*, 2010, pp. 1–6.
[3] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," *Proc. ACM IMC*, 2010, pp. 267–80.
[4] A. Tootoonchian *et al.*, "On Controller Performance in Software-Defined Networks," *Proc. USENIX Hot-ICE '12*, 2012, pp. 10–10.
[5] M. Yu *et al.*, "Scalable Flow-Based Networking with DIFANE," *Proc. ACM SIGCOMM 2010 Conf.*, 2010, pp. 351–62.
[6] A. R. Curtis *et al.*, "DevoFlow: Scaling Flow Management for High-Performance Networks," *Proc. ACM SIGCOMM '11*, 2011, pp. 254–65.
[7] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *ACM SIGACT News*, vol. 33, no. 2, 2002, pp. 51–59.
[8] A. Tootoonchian and Y. Ganjali, "Hyperflow: A Distributed Control Plane for OpenFlow," *Proc. 2010 INM Conf.*, 2010, pp. 3–3.
[9] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," *Proc. HotSDN '12 Wksp.*, 2012, pp. 19–24.
[10] M. Casado *et al.*, "Ethane: Taking Control of the Enterprise," *Proc. 2007 Conf. Applications, Technologies, Architectures, and Protocols for Computer Commun.*, 2007, pp. 1–12.
[11] N. Foster *et al.*, "Frenetic: A Network Programming Language," *Proc. ACM ICFP Conf.*, 2011, pp. 279–91.
[12] P. Kazemian, G. Varghese, and N. McKeown, "Header Space Analysis: Static Checking for Networks," *Proc. USENIX NSDI '12*, 2012, pp 9–9.
[13] N. Handigol *et al.*," Where is the Debugger for My Software-Defined Network?," *Proc. ACM HotSDN'12 Wksp.*, 2012, pp. 55–60.

Biographies

Soheil Hassas Yeganeh is a Ph.D. student in the Department of Computer Science at the University of Toronto. His research interests are in software-defined networking, network virtualization, and congestion control.

Amin Tootoonchian is a Ph.D. candidate in the Department of Computer Science at the University of Toronto, and a visiting graduate student at the University of California, Berkeley and the International Computer Science Institute. He is broadly interested in the design and implementation of networked systems, especially software-defined networks, information-centric networks, and online social networks.

Yashar Ganjali is an associate professor of computer science at the University of Toronto. His research interests include packet switching architectures/algorithms, software defined networks, congestion control, network measurements, and online social networks. He has received several awards including an Early Researcher Award, Cisco Research Award, best paper award at the Internet Measurement Conference 2008, best paper runner up at IEEE INFOCOM 2003, best demo runner up at ACM SIGCOMM 2008, and first and second prizes in the NetFPGA Design Competition 2010.