TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 01/10 pp221-232
Volume 20, Number 3, June 2015

On Scaling Software-Defined Networking in Wide-Area Networks

Shuhao Liu and Baochun Li*

Abstract: Software-Defined Networking (SDN) has emerged as a promising direction for next-generation network design. Due to its clean-slate and highly flexible design, it is believed to be the foundational principle for designing network architectures and improving their flexibility, resilience, reliability, and security. As the technology matures, research in both industry and academia has designed a considerable number of tools to scale software-defined networks, in preparation for the wide deployment in wide-area networks. In this paper, we survey the mechanisms that can be used to address the scalability issues in software-defined wide-area networks. Starting from a successful distributed system, the Domain Name System, we discuss the essential elements to make a large scale network infrastructure scalable. Then, the existing technologies proposed in the literature are reviewed in three categories: scaling out/up the data plane and scaling the control plane. We conclude with possible research directions towards scaling software-defined wide-area networks.

Key words: software-defined networking; scalability; OpenFlow

1 Introduction

Software-Defined Networking (SDN), especially OpenFlow[1]-based SDN, is believed to have the potential to revolutionize the Internet architecture towards the next generation[2]. The SDN paradigm decouples the control plane from the data plane completely, pushing all the intelligence up to the software in a logically centralized controller. Starting from the scale of a campus network[1], its deployment in Wide-Area Networks (WAN) is a promising direction in practice: Google has already deployed software-defined networking to manage their inter-datacenter networks[3].

The main advantages of SDN—the possibility of an unprecedented degree of flexibility and programmability—are crystal clear, which make monitoring and operating a network as easy as developing a software application. SDN is like the ultimate toolkit for network operators, bringing network management to a brand new era with fine-grained, dynamic, and complete control over network flows via software[4]. Specifically, there are three primary benefits brought forth by software-defined networking: (1) traffic is flexible and easy to control at the per-flow or even per-packet granularity; (2) policy changes can be completed within several milliseconds, instead of taking several minute to react[5]; and (3) control applications are easy to update when abnormal events occur in the network.

However, combining the granularity, flexibility, and performance requirements together, deploying a software-defined WAN suffers from scalability issues. For example, the TCAM space limitation problem is among the most severe ones in the data plane. State-of-the-art hardware cannot support a sufficient number of TCAM rules to enable fine-grained flow identification

• Shuhao Liu and Baochun Li are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto M5S 3G4, Canada. E-mail: {shuhao, bli}@ece.toronto.edu.

∗ To whom correspondence should be addressed.

Manuscript received: 2015-05-05; accepted: 2015-05-11

as the packet header diversity grows in WANs[6]. In the control plane, the controller can easily become the bottleneck of the network without proper physical distribution or offloading mechanisms[7].

To tackle these issues that significantly limit the scale of a software-defined WAN, both industry and academia have proposed their solutions. On one hand, existing techniques in large scale systems are still valid, but they need to address the new challenges. On the other hand, brand new, specialized designs should be adopted. The representative efforts made in the literature are surveyed in this paper, including new architectures and tricky mechanisms. Note that since the data plane and the control plane are network components of distinct domains in SDN, we examine the related works respectively.

First, we revisit the design of a distributed system that has been proved successful, the Domain Name System (DNS), trying to uncover the inherit design principles of an Internet scale infrastructure. By analysing both the similarities and the differences between a software-defined WAN and the DNS, we do not consider some designs that are no longer applicable, and instead emphasize some essential ideas that are useful.

Then, we review existing designs of scalable control planes, which have always been a heated research topic since the dawn of SDN. Essentially, the control plane is similar to any distributed system[8], hence its scalability is not fundamentally bottlenecked[9]. Existing works are proposed to scale the controller capacity not only horizontally but also vertically.

The data plane in an ideal SDN framework is described as a dumb packet-switching network with little intelligence, because the capability of making decisions has been entirely moved to the control plane[10]. Surprisingly, old-school technologies (e.g., MPLS[11,12] and commodity IP network routing) can be extended to help us out. Also, there exist some hybrid SDN solutions that are proposed to return the intelligence back to the data plane.

The rest of this paper is organized as follows. In Section 2, we provide a thorough review of the scalability challenges to scale SDN in a WAN, stating the motivation of our study. Section 4 presents our taxonomy of existing technologies. By comparing with DNS in Section 3, we summarize some important principles of a distributed system design. Then, the representative works in the literature will be discussed in detail in Sections 5-7. Our analysis and insights, as well as future research topics from our perspective, will be presented in Section 8. Finally, Section 9 concludes the paper.

2 Scalability Issues in SDN

2.1 Software-defined WANs: Requirements

To gain the same benefits as small scale software-defined networks, software-defined WANs cannot work properly by just extending a small scale SDN. There are a number of requirements for software-defined networks to scale up to a wide-area environment:

- *Global knowledge*. Configuring decisions are usually made to achieve a globally optimized state, which requires global knowledge of the network traffic[5].
- *Real-time monitoring, decision making, and policy updating*. Beyond the large volume of information, the control plane requires dynamic monitoring of the network. Whether proactively or reactively, the effective configurations should be based on real-time updated information of the network traffic. Moreover, the new policy should be computed and updated to the proper switches in real time[13,14].
- *Fine-grained control*. An ideal SDN system should be capable of managing the traffic at the per-flow or even per-packet level due to the rich diversity of QoS requirements. To maximize user experience, fine-grained control over flows (or packets) should be an important objective in an SDN-based WAN.
- *Make the right decision, perform the right change*. It is another great challenge for the control plane to update network policies in a consistent manner, that is, to guarantee the *atomicity* of changes.

To meet these requirements, such a network architecture would face great challenges at large scales, both in the control plane and in the data plane.

2.2 Control plane scalability

The first two requirements summarized in Section 2.1 are crucial for control plane designs in the SDN paradigm.

The demand of global knowledge is much harder to satisfy. As the network scales out, the minimum size of monitoring messages will grow rapidly ($O(N^2)$, where N is the diameter of the network). Also, the number of message sources will become larger, making it much harder to incorporate with.

Unfortunately, the demand for real-time information update makes things worse. The challenge here is two-fold. First, aggregating and disseminating large quantities of information in real time may easily congest the controller. Second, optimizing the configuration decisions usually involves solving linear problems[5, 15], which is quite demanding in computation power.

Since a single node has both limited computation and storage capacities, it is impossible to accomplish the aforementioned objectives by solely using a higher-end controller. Therefore, scaling out as well as scaling up mechanisms should be designed properly. Scaling out technologies are similar to those in a typical distributed system.

2.3 Data plane scalability

The last two requirements presented in Section 2.1 are translated into scalability challenges in the data plane.

In practice, it is difficult to treat a large number of flows differently by setting their own unique policies; therefore, network operators usually compromise to support only a limited number of service classes (e.g., SWAN[15] supports three classes, i.e., interactive, elastic, and background). Fine-grained management is hard to achieve because the TCAM space in an OpenFlow switch fails to scale up. Undoubtedly, the ideal fine-grained flow differentiation will lead to the generation of many more TCAM rules. However, TCAM is prohibitively expensive: only a few thousands of TCAM rules are supported by even the next-generation OpenFlow switches[16]. Moreover, the large network scale further aggravates the situation. The number of concurrent flows passing through a switch (especially a core switch) will grow as the number of end hosts increases, which will typically make the TCAM space more scarce.

As for atomic network upgrading, according to Refs. [17, 18], massive packet losses and stochastic states of the network will occur without proper mechanisms to ensure the per-packet consistency during policy updates. In addition to incorrect routing, the overhead of update messages may increase linearly as the network scales[19]. This degree of uncertainty is undesirable in software-defined wide-area networks.

3 DNS: A Comparison Study

As one of the most common and the most successful distributed systems that are widely deployed in the Internet, the DNS is a useful analogue of the control plane in software-defined WANs to tackle scalability issues. Indeed, it is reasonable to borrow some ideas from the DNS design to power a distributed control plane. However, though sharing some basic characteristics as a system that is both geographically and functionally distributed, differences of inner functioning mechanisms do make them distinct in practice. In this section, we try to go through the similarities and differences between them, summarizing what experience we can learn from the past practice.

3.1 DNS

The DNS provides name service to the Internet, that is, translating a domain name, which is a unique variable-length string identifying a network resource, to its corresponding IP address(es)[20]. The use of domain names decouples network resources from their IP addresses, providing more flexibility towards address updates and simplifying user-side addressing. Since DNS should be highly available, be quick to respond and update in time to serve all network users dynamically across the Internet, its scalability was a major challenge as the Internet grew at an unprecedented speed.

Thanks to the highly hierarchical assignment of domain names, DNS can be easily organized into a corresponding hierarchical manner[21], both in the context of geographical and functionality management. For example, a typical domain name is `xxx.yyy.zzz`. `zzz` is the top-level domain (e.g., prominent domains such as `com`, `net`, and `edu`, country code top-level domains such as `ca`, `cn`). Below the top-level domains, the second and third-level domains are available for users' reservation, enabling their local area networks to be accessible from a name that is easy to remember. More importantly, these domain names are usually under *full supervision of domain name registrars*.

The design of domain names makes the DNS horizontally scalable in nature, because a distributed database that serves hierarchical reference queries only[22]. Moreover, usage of cache in each level of DNS greatly degrades the necessary communications in one single resolving transaction. An example of a typical DNS query chain is depicted in Fig. 1, which illustrates the caching and recursive searching mechanisms that are key technologies in the DNS. Note that with the query transaction shown in the figure, the resolving results for `miami.csl.toronto.edu` will be available in all three caches in the figure for a

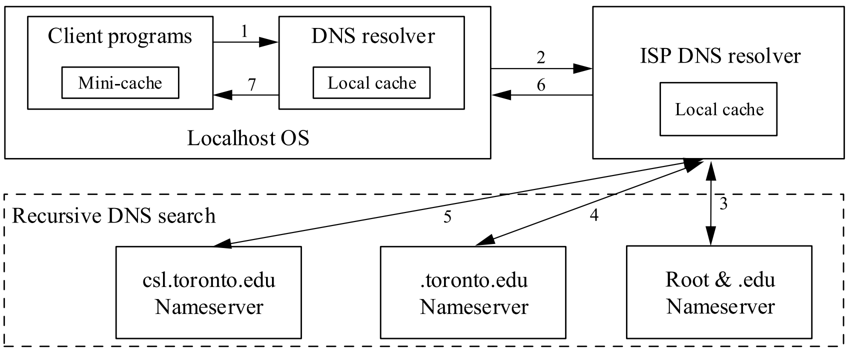

Fig. 1 An example of multiple-tier caching and recursive DNS search for a name query `miami.csl.toronto.edu`. If client programs cannot find the requested name in its mini-cache or (1) in the local host DNS resolver cache, it will (2) resort to the ISP DNS resolver. If it is not resolved either, recursive DNS search will be executed, according to the query domain name structure. First, (3) ISP asks *root* (which serves `edu` namespace) for a `toronto.edu` server. Then, (4) ISP asks `toronto.edu` about `csl.toronto.edu`. Finally, (5) `csl.toronto.edu` is found and the query is resolved, whose result IP address will then be (6,7) returned to the client.

time period.

Generally speaking, it is the caching and structured domain names that simplify the DNS. Quite a few queries should be made to effectively refine the large global domain namespace.

3.2 Similarities

As distributed systems with a logically centralized "brain", both the DNS and software-defined WANs have some common design objectives as the repository for information. The DNS is fundamentally a large database, where responses are database look-up queries, whereas the control plane is a bit more complex.

Transaction patterns. Both systems function in a *query-response* pattern. Each transaction in the system is triggered by a query that can be regarded as an independent event. In the DNS, such a query is an URL or host name to resolve. In SDN, it is a sample packet that needs to be dealt with by a switch, according to specific policies or rules. The transactions do not have strong connections with each other—queries are not assumed to be dependent—so that transactions can be programmed or analysed in a per-query basis.

Need for availability and reliability. Both the DNS server and the controller in a software-defined WAN should be an infrastructure that is highly available and reliable at any time. Failure on these infrastructures is catastrophic because it destroys either reachability or connectivity among network nodes. Therefore, these two systems should take measures on ensuring robustness.

Need for consistency and atomicity. There may exist updates in the network at any time. For example, once a web server has migrated from one IP address to another, its corresponding information should be updated in time among all relevant DNS servers, or the service provided on the host will be unavailable for some users. On the other hand, in SDN, policy changes for a single flow should be consistently applied to all affected switches in a short enough time period, otherwise packets will be lost during network updates.

Stringent requirements on response time and correctness. The latency and imperfect responses in both systems directly translate into degradation on user experience. For example, delays of a DNS server or a controller will induce the same amount of delays at the user-level, resulting in similar retrying mechanisms that may be a further burden to the system.

3.3 Differences

Despite the similarities above, it is the distinctions between two systems make traditional technologies in the DNS less useful in the context of software-defined WANs. A scalable design of the control plane in software-defined WAN is much more complicated when the following aspects are considered.

Much higher degree of dynamics. In the DNS, the matches between host names and IP addresses are basically static, because the address bindings are unchanged for most of the time. However, it is common that the routing or QoS policies vary on a per-flow basis in SDN. In extreme cases, packets in a single flow may be applied several different rules during its lifetime. For this reason, the caching mechanism, which greatly degrade the DNS complexity, is much less effective in SDN. Though we still regard TCAM rules in the

OpenFlow switches as local caches, it is quite likely that switches still need to consult the centralized controller for each new flow.

Less structural queries. As was discussed above, DNS queries are highly structural—the host names or URLs are all originally hierarchical—so that the namespaces are easy to be partitioned as well as to be distributed. Unfortunately, the control plane in SDN process packets based on the packet headers, which contain service, protocol, address, and even connection identifiers. The field space of packet headers seems not organized at all. The diversity for packets in WAN is too high to be properly partitioned by a common set of rules; therefore, it is not as easy as the DNS to apply divide-and-conquer strategies.

Demand for global knowledge. In addition to per-flow rule management, another challenge for software-defined WAN is that the decisions are made based on global knowledge. In contrast, the DNS is just a look-up process that is free from decision-making and background information. All a DNS server needs to know is the address of a host name item, such that any DNS server with a single match stored in its database is able to respond correctly. However, global knowledge is required by a considerable number of network flows in a wide-area network to ensure global optimality. Moreover, the control plane in SDN requires bi-directional information update, that is, the centralized controller is always retrieving status information (e.g., link failures and abnormal packet corruptions) from the data plane to make adaptive decisions. The demand for global knowledge makes it nearly impossible to exploit partitioning.

3.4 Experience learnt from DNS

The success of the DNS benefits significantly from the use of caching and hierarchical structuring of names[23]. Unfortunately, according to our preceding analysis, the direct use of these two mechanisms is far less beneficial in a software-defined wide-area network. Nevertheless, the brilliance in the design of the DNS may still motivate new designs for software-defined networking to scale up to the wide area.

Exploiting locality. Although caching is less efficient in SDN, locality is still the most important tool to improve scalability. Exploiting locality at scale is the key to low latency and high availability.

Artificial hierarchy. Indeed, the diversity of packet headers and the lack of relevance information makes the problem space unlikely to be partitioned. But the encapsulation techniques and the flexibility to modify packets are still optional. With artificial identifiers in the packet header, intelligence can be taken to rebuild the hierarchical architecture of the flow configuration and administration.

Replication, synchronization, and fault tolerance. As the performance of a single node cannot scale vertically free from bound, the usage of multiple replications in the network is one of the essential ideas to build a reliable, scalable system. Not surprisingly, such technologies are mature enough and already available in the SDN controllers, including answers to fault tolerance, replication, and synchronization among nodes.

4 Overview of the Available Technologies

In the past several years, a large number of research papers have been published to push SDN technologies towards the goal of wide deployment in the Internet. Scalability is among the most interesting topics, since it was intuitively recognized as an inherent problem of the SDN architecture.

To emphasize the idea of existing works, the TCAM spaces are conceptually taken as the caches in the data plane. To scale out the control plane in SDN to meet the operator requirements, fundamental principles are successfully borrowed from distributed system designs, including hierarchical solutions[7] and partition solutions[24]. On the other hand, sticking to the one centralized controller architecture, cluster operating systems[8] for OpenFlow controllers are developed to increase the single controller capacity by several orders of magnitude.

In the data plane, there are basically two categories of proposals*. First, we may leverage label switching (e.g., MPLS[11,12]) to offload the network core. In Ref. [25], Casado et al. argued that it is reasonable for future software-defined WAN to have an intelligent edge and a label-switched (typically MPLS-based) core. We think this argument is valid because it is a simple but scalable architecture. The other line of work manages to propose a hybrid solution. Synthesizing SDN control

* Note that proposals on TCAM space optimization and multiple flow tables in hardware, in a sense, do scale up the single switch capacity, but the methodology is indirect. These works are beyond the scope of this paper, which focus solely on the technologies that can be employed to directly leverage TE scalability.

with the traditional distributed network protocols, such as BGP, OSPF, and IS-IS, is a practical architecture to scale. An intelligent integration makes the scheme capable of gaining flexibility from SDN, while still benefiting from the distributed architecture to ensure scalability[26].

5 Scaling Out the Control Plane

In this section, we examine the techniques in the literature to scale the control plane horizontally. In this line of works, making a large system hierarchical and/or partitioning the workload among several workers remain to be the essential ingredients to be more scalable. As has been discussed in Section 3.4, locality should also be used in software-defined wide-area networks. In the early stages of software-defined networking, the proposals we will now show have explored solutions that inherit from the heritage of the Domain Name System.

5.1 DIFANE

DIFANE[6] is mainly a partitioning scheme to offload the TCAM space pressure in the data plane. The usage of *authority switches* is the key in the DIFANE architecture. As shown in Fig. 2, the network is partitioned into several pieces depending on the scale, with each piece managed by one authority switch. The controller will *proactively* interact with the authority switches, hoping that the rules can be installed *before* they become active. At the same time, the logically centralized controller should manage network partition information besides generating flow configuration rules as traditional controllers. It continuously updates the network/traffic partitioning information to the switches on the data path as well as the authority switches. In this way, the administration pressure from un-hit flows in the data path is able to be partitioned and balanced effectively among authority switches, thus scaling the control plane across the network.

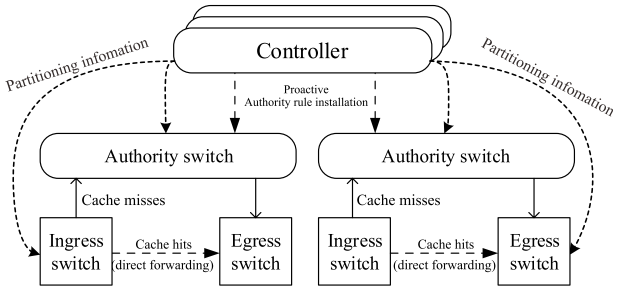

Fig. 2 DIFANE[6] architecture.

The authority switches provide an additional level of flow rule caches, making the conceptual caching mechanism more efficient. Moreover, DIFANE[6] suggests that though the controllers should *generate* the configured rules, the real-time handling of packets should not be their responsibility.

DIFANE is literally an abbreviation for “DIstributed Flow Architecture for Networked Enterprises”. Though the name suggests that the original design is for the scale of an enterprise network, the architectural design has the potential of facing several higher orders of magnitudes of load in wide-area networks. However, we think the assumption of rules can be computed before the arrival of actual sample packets is too strong to be held in wide-area networks. The demand for proactive rule installation (and thus caching) is not applicable for many reactive applications, such as streaming media traffic engineering where flows should be reactively configured without resource reservation beforehand.

5.2 HyperFlow

HyperFlow[24] improves the scalability of software-defined networks by partitioning the network to several physically distributed controllers. A relatively large scale network, say an autonomous system, is basically a collection of interconnected sub-networks. Each sub-network is then able to be controlled by a controller, and we assume that the connections among sub-networks have bandwidth reserved for control messages. HyperFlow is proposed to actively synchronize the global view among the controllers disseminated in all sub-networks. Therefore, each controller will be able to make decisions without reactively consulting others to retrieve relevant information.

Moreover, a trick to make this idea possible is that the synchronization process among controllers is event-based, with information published selectively through a publish/subscribe system. Based on the observations that network-wide changes (in the form of emitted events) which need to be broadcasted are relatively rare, and that network applications only require minimal updates from these events, HyperFlow scales well with one constraint—the delays between the farthest pair of controllers in the network—that directly affects the state consistency.

It is highly likely that the assumptions made in this work will hold in wide-area networks. Indeed, HyperFlow is effective with a proper partition of the network, which does not seem a complex task even in a generic WAN environment. There are two possible concerns beyond these assumptions.

One is that, despite the relatively low rate of event occurrences that should be published, the number of events still grows linearly as the number of network nodes and links increases. For this reason, the bandwidth reserved for controller synchronization and the network scale is still limited by the achievable performance of a single controller.

The other concern is about the network consistency, because the worse-case delay in the network increases as the network scales. The consequences of these inevitable delays should be further investigated.

5.3 DevoFlow

DevoFlow[27], on the contrary, tries to tackle scalability issues theoretically by discussing the understanding of the SDN principle. *Is it enough to have partially decoupled control plane?* To make the SDN design available at a larger scale or survivable with the need for high performance networking, DevoFlow tries every means to keep flows in the data plan. After analysing the network application in SDN, one of the key conclusions is that flows which are not "significant" can impact little when the control plane is making decisions. Based on this observation, DevoFlow enables less-important flows to be processed directly in the data plane. Therefore, the accessible information and visibility of the control plane is reduced with little cost in decision making. Furthermore, such level of vague processing in the data plane will lead to radical use of wildcard rules, so fewer TCAM rules are needed and handware cost will be reduced. Above all, most flows are pushed back to the data plane to be processed, and the statistic collected from the controller is reasonably limited.

Obviously, DevoFlow is scalable in nature: it tackles the fundamental issues of scalability. However, questions arise on the flexibility of the design. Filtering less-useful information at the data plane has the risk of losing visibility and flexibility of the centralized control, because it is hard to decide whether a specific set of information should be filtered. The answer is likely to depend on the running network applications. However, at this stage of SDN, it is quite difficult to predict the characteristics of future network applications.

5.4 Kandoo

Kandoo[7] proposes a highly configurable control plane architecture with a two-tier (see Fig. 3) or even more tiers hierarchy structure. The insight of this work is to categorize switches into several tiers, so as to authorize their level of "local" events. For local applications that require only local information and network updates, it can be handled at the lower-tier switches; on the contrary, the message will be processed and delivered to its ancestor switch. Specifically, in order to fully exploit locality, the switches at the bottom tier, namely the local switches, are placed as close to the data plane as possible. For example, an extreme implementation would be a hardware component attached to switch chips.

Conceptually speaking, lower-tier controllers are employed as "intelligent" caches to shield a reasonable amount of work load from the upper-tier controllers, except that the caches can make its own decisions based on its authorized information. For example, an OpenFlow application that can detect large flows in the network does not need any global information. Even the information from a single switch is enough. Then, some applications, such as the ones making routing decisions, can retrieve the *result* of these local applications without details, thus to off-load the root controllers. The proposed evaluation suggests that Kandoo be able to improve the control plane capacity by over one order of magnitude.

Kandoo[7] represents the attempts to exploit locality in SDN. We should be aware of that Kandoo is evaluated under the assumption that there exist two sets of applications in the network: one is local and the other is global applications, so that authority can be partitioned smoothly between layers of controllers. In reality, a software-defined WAN architecture must be capable of holding a large number of sets of network applications rather than a carefully defined set. For example, it is highly likely that most applications in the network requires either global information or

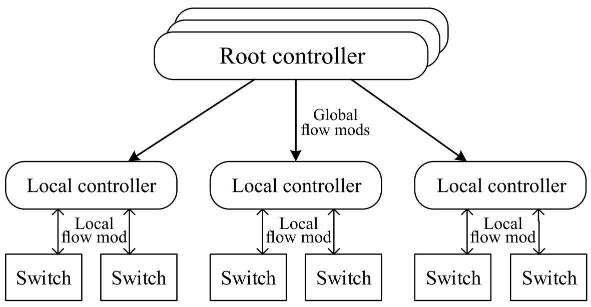

Fig. 3 An example of the two-tier Kandoo[7] architecture.

global configuration. In these cases, Kandoo will fail to provide scalability, which may introduce control plane overhead as the packets should be processed by a sequence of controllers in multiple tiers until having the access to enough information.

The inspiration from Kandoo is that, given the supported application characteristics in WAN, we can design dedicated architectures to scale it out. It is not a good assumption to hold, because such an infrastructure architecture design may reversely limit the development of network applications, which is a violation of the SDN principle.

5.5 Discussions

In this section, four scalable designs are reviewed: DIFANE[6], HyperFlow[24], DevoFlow[27], and Kandoo[7]. Generally speaking, they are representative works from different aspects. HyperFlow tries to reasonably partition the network to perform divide-and-conquer. DevoFlow is proposed to filter information before it is transmitted up to the controller, thus to offload the bottleneck. DIFANE and Kandoo are similar hierarchical solutions with conceptual distinctions. DIFANE pays more attention to create multiple levels of rule caches, while the latter one exploits locality of certain network applications.

It is easy to see that all these attempts still follow the design principles of typical distributed systems (Section 3.4), especially in exploiting locality. However, as a clean-slate networking technology, few people can be confident enough to predict the future trend of network applications. To be honest, we think that our community is still far from the goal of designing a generic architecture for software-defined WAN without assumptions on application behaviours.

To that end, a promising research and industry direction lies in traffic monitoring and measurement results in a production software-defined networks.

6 Scaling Up the Control Plane

To improve scalability further, it is also important to "scale up" the performance of a single controller within the control plane. We are now ready to discuss a number of proposals towards this direction, which is orthogonal as compared to the previous section.

Onix[8] is an operating system designed to support a cluster of controllers. Onix itself supports state synchronizing and distributed computing schemes among the controller nodes in the same cluster. Being physically co-located, the cluster of controllers can maintain a network-wide view with scalability and reliability. Instead of depending on a single controller instance, a cluster of controllers provides both capacity and robustness, while its internal design is transparent to the data plane, i.e., an Onix works like a single controller from the switches' perspectives. The typical architecture of an Onix cluster is depicted in Fig. 4.

Inside an Onix cluster, the workload on a single controller instance is reduced because all queries from the data plane are partitioned to the administration of multiple homogeneous nodes by the underlying switch management components. All these nodes are sharing the global network information in a distributed database. Each node exposes a subset of its administrating information, while replication and restoration mechanisms are employed just as a traditional distributed database system. The network within the cluster is highly available and dedicated to the updating and synchronizing traffic among nodes, such that network information that is stored on another node can be accessed with small enough overheads.

In this design, the computation and storage capacity of a single controller is multiplied as the number of nodes in the cluster increases, which is ideal for a component controller for a large scale software-defined network. Furthermore, clusters of Onix can still be organized in an architecture listed in Section 5, and the scalability can be further improved.

Provided that the network statistics/state information in a WAN-scale SDN is extremely large in size, a cluster as a single controller will be the only resort.

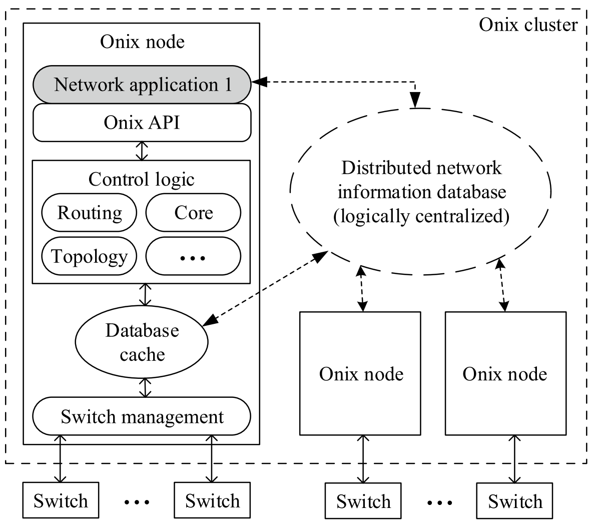

Fig. 4 A typical architecture of an Onix[8] cluster.

7 Scaling the Data Plane

The data plane should be fundamentally "dumb" in the SDN architecture, which performs packet forwarding based on controller instructions solely. The instruction is either queried from the controller or cached in the TCAM. The former action will suffer from the query delay, so the possibility of hitting the cache in TCAM is critical for network performance, especially in the network core at scale.

7.1 Label-switching to offload the core

Label switching schemes are one of the valuable heritages from tradition traffic engineering technologies[4]. It moves the intelligence and complexity to the network edges, offloading the network core.

As depicted in Fig. 5, label switching effectively decouples the network edges from the core. The edge switches in the network, will fully handle its corresponding network flows, making routing decisions and packet scheduling. On the contrary, the operator manages to achieve a "dumb" network core. This model is both valid and practical because it relieves the system bottleneck. In a large-scale network, say the Internet, the core switches have to handle a much greater orders of magnitude of flows, which is typically translated into a great number of TCAM rules. As the TCAM spaces fail to scale up, the core becomes the bottleneck. With label switching, the network core is able to get rid of the complex and expensive TCAM. Instead, some fixed-length, exact match labels that are better available in commodity hardware are used to perform switching[28]. The label switching rules are capable of being pre-installed. Also, the network edge, which is naturally less demanding in TCAM because of the small set of passing flows, still exploits the flexibility of the SDN paradigm.

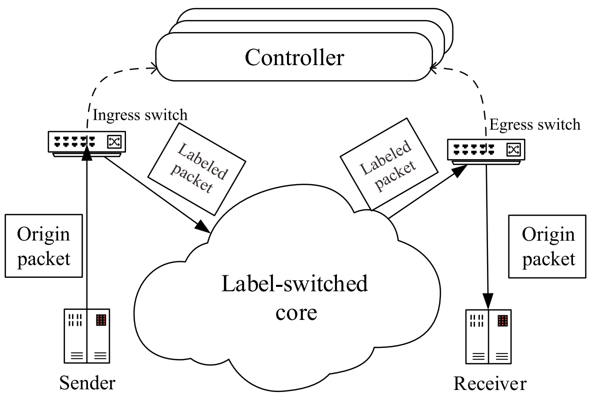

Fig. 5 The concept of lable switching in SDN.

At the same time, label switching happens to bypass the consistency issues of network update in a large scale software-defined WAN. Since the edge switch is the single node involved in the update, per-packet consistency is guaranteed. Therefore, we believe label switching could be the right solution to tackle the data plane scalability issues in one action.

7.1.1 MPLS-based label switching in SDN

Fabric[25] is a representative work that introduces MPLS-based label switching in the software-defined WAN. Since the support for MPLS standard is readily available in commodity switches, implementing MPLS-based labelling is an effort with little cost overhead. The Fabric architecture is a bit different from the one depicted in Fig. 5. A "Fabric controller" is employed to control the MPLS-based core to offer more flexibility and dynamics. In order to match the edge context to the edge, which is essentially a simple namespace translation, the encapsulation technology is used.

Casado et al.[25] argued that this architecture, with separation of forwarding and control provided by the SDN principle and the simplicity provided by MPLS, can benefit from both technologies.

7.1.2 Other label switching in SDN

Shadow MACs[19] is a recent proposal that utilizes a virtual MAC address to perform label switching in SDN. Rather than choosing a dedicated label (e.g., MPLS), Agarwal et al.[19] successfully found a means to implement this concept with commodity SDN hardware that has already been deployed. In fact, one of the interesting highlights of this work is that it can be implemented in two ways: one is letting the edge switches perform virtual MAC rewriting (reverting) guided by OpenFlow instructions; the other is to exploit ARP spoofing technology to making the sender to use the virtual destination MAC address automatically.

Shadow MACs requires no change in hardware, not even support for OpenFlow. For this reason, it is highly practical for the sake of deployment. However, due to the process of rewriting hardware addresses, the hybrid use of shadow MACs and real MAC addresses is not safe. Hardware address conflicts are not easy to avoid.

7.2 Hybrid solutions

Due to the distributed nature of traditional routing schemes, the WAN is scalable. The SDN paradigm centralizes network functionalities, but is not orthogonal to the traditional network management. Some hybrid schemes accomplish to exploiting flexibility from SDN, but keep up with the scalability of distributed routing protocols at the same time.

Fibbing[26] makes the controller to trick the OpenFlow switches by depicting a fake topology, thus to let the conventional OSPF perform desired flexible routing, bypassing the network inconsistency and TCAM space unavailability. By generating fake messages that can be identified and used by the OSPF components in traditional switches, the switches will then make routing decision on their own based on the execution results of OSPF. With this little trick, the switches do not have to cache TCAM rules, and the switching performance remains unaffected.

RFCP[29] attempts to build a centralized routing control platform with the help of OpenFlow, enabling AS-wide BGP routing service while maintaining full control over the network. This work is similar to Ref. [26].

Since the above hybrid solutions completely rely on the integration with traditional distributed network protocols, only can the corresponding network functionalities be supported. Also, it does not improve the network programmability—there is no way to simplify the hybrid application programming, nor to reuse codes among different functionalities. Above all, hybrid solutions can be regarded as trade-offs to gain higher level of control. They should not be the final answer to scalability issues in a software-defined WAN.

8 Discussion: Violation of the SDN Principles

As introduced in Section 3.4, the design of a distributed system at WAN scale should follow three essential principles. However, all these principles are, in a sense, violation to the SDN principle, i.e., completely decoupling the control plane from the data plane.

We argue that it is an inevitable solution to push some control functions downward, closer to the data plane to exploit locality and to reduce response time. Note that distributing some control logic closer to the data path does not necessary imply violation of the clean separation between them. Essentially, it is *caching control logic closer to the datapath*. Controller still has the ability to handle every related events and to make the decision.

Caching control logic on switches is not a new idea. In fact, the latest version of OpenFlow supports simple logics in flow table[1]. To be more specific, conditional rules are optional, such that the control decisions can perform local fast failover[30,31] in case of sudden link failure. Further, Ref. [32] theoretically proves that under such a standardized control scheme, the potential computation capacity in the dataplane is enough for flexible control.

Therefore, caching logic instead of static rules in the data plane seems to be a good solution to the scalability problem.

9 Conclusions

Software-defined networking suffers from severe scalability issues towards deployment in larger scale networks such as wide-area networks. The challenges are three-fold:

- A centralized control plane has to store, process, and maintain an unbounded amount of network state information at scale. Moreover, an unbounded number of events emitted should be handled in real time.
- Since a wide-area network spans a large area geographically, the centralized control plane should consider the propagation delays which may significantly degrade the response time.
- The relatively high diversity of OpenFlow matching rules is a double-edged sword. It may result in a huge number of TCAM rules, which are not cost-effective to be cached in the data plane.

We have examined a few early attempts in the research community to address these scalability problems. Generally speaking, these surveyed proposals are learning from the experience of existing large distributed system designs such as DNS. The principles are shared: exploiting locality, hierarchical structuring, replication and synchronization among the nodes. Although solutions to software-defined wide-area networks rely on more practical experiences, we think more attention should be paid to thinking about the foundational design principles of software-defined networking.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner,

Openflow: Enabling innovation in campus networks, *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 69–74, 2008.

[2] R. Jain and S. Paul, Network virtualization and software defined networking for cloud computing: A survey, *IEEE Communications Magazine*, vol. 51, no. 11, pp. 24–31, 2013.

[3] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al., B4: Experience with a globally-deployed software defined wan, in *Proc. of ACM SIGCOMM*, 2013.

[4] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, A roadmap for traffic engineering in sdn-openflow networks, *Computer Networks*, vol. 71, pp. 1–30, 2014.

[5] S. Agarwal, M. Kodialam, and T. Lakshman, Traffic engineering in software defined networks, in *Proc. of IEEE INFOCOM*, 2013.

[6] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, Scalable flow-based networking with difane, *ACM SIGCOMM CCR*, vol. 40, no. 4, pp. 351–362, 2010.

[7] S. Hassas Yeganeh and Y. Ganjali, Kandoo: A framework for efficient and scalable offloading of control applications, in *Proc. of ACM SIGCOMM HotSDN*, 2012.

[8] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al., Onix: A distributed control platform for large-scale production networks, in *Proc. of USENIX OSDI*, 2010.

[9] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, On scalability of software-defined networking, *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013.

[10] M. Kobayashi, S. Seetharaman, G. Parulkar, G. Appenzeller, J. Little, J. Van Reijendam, P. Weissmann, and N. McKeown, Maturing of openflow and software-defined networking through deployments, *Computer Networks*, vol. 61, pp. 151–175, 2014.

[11] D. O. Awduche and J. Agogbua, Requirements for traffic engineering over mpls, RFC 2702, Tech. Rep., September 1999.

[12] D. O. Awduche, Mpls and traffic engineering in ip networks, *IEEE Communications Magazine*, vol. 37, no. 12, pp. 42–47, 1999.

[13] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, Oflops: An open framework for openflow switch evaluation, in *PAM*, 2012.

[14] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca, Planck: Millisecond-scale monitoring and control for commodity networks, in *Proc. of ACM SIGCOMM*, 2014.

[15] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, Achieving high utilization with software-driven wan, in *Proc. of ACM SIGCOMM*, 2013.

[16] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, Past: Scalable ethernet for data centers, in *Proc. of ACM CoNext*, 2012.

[17] R. Mahajan and R. Wattenhofer, On consistent updates in software defined networks, in *Proc. of ACM HotNets*, 2013.

[18] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, Abstractions for network update, in *Proc. of ACM SIGCOMM*, 2012.

[19] K. Agarwal, C. Dixon, E. Rozner, and J. Carter, Shadow macs: Scalable label-switching for commodity ethernet, in *Proc. of ACM SIGCOMM Workshop HotSDN*, 2014.

[20] P. Mockapetris, Rfc 882: Domain names: Concepts and facilities, november 1, 1983, Obsoleted by RFC1034, RFC1035 [Moc87b, Moc87c]. Updated by RFC0973 [Moc86]., 1986.

[21] P. V. Mockapetris, Rfc1035: Domain names—implementation specification, 1987.

[22] P. B. Danzig, K. Obraczka, and A. Kumar, An analysis of wide-area name server traffic: A study of the internet domain name system, in *Proc. of ACM SIGCOMM*, 1992.

[23] P. Mockapetris and K. J. Dunlap, Development of the domain name system, in *Proc. of ACM SIGCOMM*, 1988.

[24] A. Tootoonchian and Y. Ganjali, Hyperflow: A distributed control plane for openflow, in *Proc. of USENIX INM Conference*, 2010.

[25] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, Fabric: A retrospective on evolving sdn, in *Proc. of ACM SIGCOMM Workshop HotSDN*, 2012.

[26] S. Vissicchio, L. Vanbever, and J. Rexford, Sweet little lies: Fake topologies for flexible routing, in *Proc. of ACM SIGCOMM Workshop HotSDN*, 2014.

[27] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, Devoflow: Scaling flow management for high-performance networks, in *Proc. of ACM SIGCOMM*, 2011.

[28] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, Software-defined internet architecture: Decoupling architecture from infrastructure, in *Proc. of ACM SIGCOMM Workshop HotSDN*, 2012.

[29] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. Raszuk, Revisiting routing control platforms with the eyes and muscles of software-defined networking, in *Proc. of ACM SIGCOMM Workshop HotSDN*, 2012.

[30] M. Borokhovich, L. Schiff, and S. Schmid, Provable data plane connectivity with local fast failover: Introducing openflow graph algorithms, in *Proc. of ACM SIGCOMM Workshop HotSDN*, 2014.

[31] L. Schiff, M. Borokhovich, and S. Schmid, Reclaiming the brain: Useful openflow functions in the data plane, in *Proc. of ACM HotNets*, 2014.

[32] C. Newport and W. Zhou, The (surprising) computational power of the sdn data plane, in *Proc. of IEEE INFOCOM*, 2015.

Baochun Li received the BEng degree from Tsinghua University, China, in 1995 and the MS and PhD degrees from University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000, respectively. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a professor. He holds the Nortel Networks Junior Chair in Network Architecture and Services from October 2003 to June 2005, and the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include large-scale distributed systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. Dr. Li has co-authored more than 280 research papers, with a total of over 13 000 citations. He was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the IEEE Communications Society, and a recipient of the University of Toronto McLean Award. He is a member of ACM and a Fellow of IEEE.

Shuhao Liu received his BE degree from Tsinghua University, Beijing, China, in 2012 and the MS degree from National University of Defense Technology, Changsha, Hunan, China in 2014. He is currently a PhD student at University of Toronto. His research interests include software-defined networking, datacenter networking, and network virtualization.