Design Considerations for Managing Wide Area Software Defined Networks

Reaz Ahmed and Raouf Boutaba

The authors are with the University of Waterloo.

ABSTRACT

SDN has the potential to simplify network configuration and reduce management complexity. In today's networks control and forwarding functions are tightly coupled and embedded within each switch/router. SDN, in contrast, accumulates the control functionality in one or more dedicated network entities called controllers, which provide a unified interface to configure and control the network. Packet forwarding, on the other hand, remains the responsibility of the switches/routers. Many datacenter networks have benefited from the abstraction provided by SDN. However, in a Wide Area Network (WAN) a single controller becomes a performance bottleneck. Multiple controller solutions are proposed as a natural consequence. In this article we present the requirements, design alternatives, and a possible management architecture for a single administrative domain WA-SDN. We also discuss the functional components that should be present in a multi-controller architecture for managing WA-SDN deployments.

INTRODUCTION

Managing the Internet's core has become very complex, tedious, and error-prone due to a number of reasons. The first reason is the tremendous growth in the Internet user population and the voluminous network traffic they generate. From December 1995 to March 2013 the global population of Internet users rose from 16 million to 2.7 billion.[1] In order to cope with this rapid growth, network operators have to relentlessly deploy and manage a wide variety of switches, routers, and middle-boxes (e.g. firewalls, network address translators, and load balancers). Second, these networking devices are manufactured by different vendors and have diverse configuration protocols. This heterogeneity contributes an additional level to the network management complexity.

Traditional switches/routers have two responsibilities:
- *Decide* the next hop for each packet.
- *Forward* the packet accordingly.

In contrast, Software Defined Networking (SDN) decouples decision making from packet forwarding. In the SDN paradigm routing decisions are taken by a logically centralized entity called the *controller*. The controller updates the routing table entries in switches/routers using a standard communication protocol like OpenFlow [1], while the switches blindly follow their routing table entries for forwarding packets.

SDN has emerged as a promising solution to simplify network configuration and management complexities [2, 3]. The SDN technology has been adopted in many datacenter networks. These deployments rely on a single controller for internal traffic management and tenant isolation. However, in a WAN the controller can be many hops away from the switches, and the communication latency can be much higher than that in a datacenter network. Hence, a single controller may not be efficient in a WAN [4]. This fact has been identified by a number of research works including [5–7], which propose multi-controller SDN solutions. Google's B4 network [8], the largest SDN deployment, uses multiple controllers as well. These solutions either propose a two-level controller hierarchy or a distributed controller network. In a two-level hierarchy the upper level controller becomes a performance bottleneck. On the other hand, in a purely distributed solution inter-controller synchronization overhead becomes a major concern.

In this article we first present an overview of the SDN technology along with its promised benefits. We then highlight the shortcomings of a single-controller architecture in a WA-SDN deployment, followed by the design considerations for a multi-controller SDN architecture. We build-upon the concepts established by the network management community over two decades ago, such as management by delegation and hierarchical management [9], and derive a two-level management architecture for a single administrative domain WA-SDN, where the high-level management tasks are performed through delegation of executable control tasks to lower-level managers. We discuss how these concepts can benefit future SDN deployments in WANs. Finally we present a qualitative comparison of the existing WA-SDN architectures with the presented architecture.

SDN OVERVIEW

SDN ARCHITECTURE

Figure 1 presents the components of a basic SDN solution, where OpenFlow has been used as the communication protocol between data and control planes. Here, switches and routers com-

[1] http://www.internet-worldstats.com/emarketing.htm

prise the *data plane*, while the *control plane* consists of a logically centralized controller. Data plane devices maintain a secure TCP connection to the controller, and communicate using the OpenFlow protocol. An OpenFlow-enabled switch maintains a list of packet forwarding rules in a *flow table*. A flow table entry is of the form "if *condition* then *action-list*." A condition contains packet header fields like MAC addresses, IP addresses, and port numbers for source and destination. An action-list contains actions like *modify packet header*, *forward to switch-port*, *send to controller*, *drop packet*, and so on. A switch matches the values in an incoming packet header against the condition in each flow table entry. If a condition matches, the corresponding action-list is executed. Otherwise, a Packet_In message (along with the packet header) is sent to the controller. Upon receiving a new packet, the controller decides a suitable path for the packet and installs forwarding rules in each switch along the path. Subsequent packets with the same source and destination signature will follow the same path without the controller's intervention. The controller runs in a general purpose compute hardware, and acts as a middleware for the network applications (e.g. firewall, intrusion detection, and deep packet inspection) that need to perform intelligent packet processing.

Benefits of SDN

SDN has gained much attention from both the industry and academia in recent years. This surge of attention can be attributed to the promising benefits of SDN. Here we highlight some of these benefits:

- SDN switches are simpler and cheaper as they have to implement fewer protocols, and do not need to make complex decisions.
- SDN allows independent evolution of the data and control plane elements as long as they adhere to a standard interface (e.g. OpenFlow). This loose coupling will allow different parties to independently develop interoperable control logic and management applications.
- Network applications can be easily developed and deployed on top of the logically centralized controller. On the contrary, a traditional network application may need to change the control logic in a switch firmware.
- SDN offers vendor neutrality by hiding the vendor specific implementation details behind a standard interface between switches and the controller.
- Network management should be simpler and less error-prone in SDN. In contrast to controlling and configuring each switch independently, a network administrator can program the controller, which in turn can configure the switches as needed.

Shortcomings of SDN

Despite numerous benefits, the networking industry has not yet seen widespread adoption of the SDN technology in WANs. Though it is difficult to identify the exact reasons behind this restraint, we highlight some of the technical challenges here:

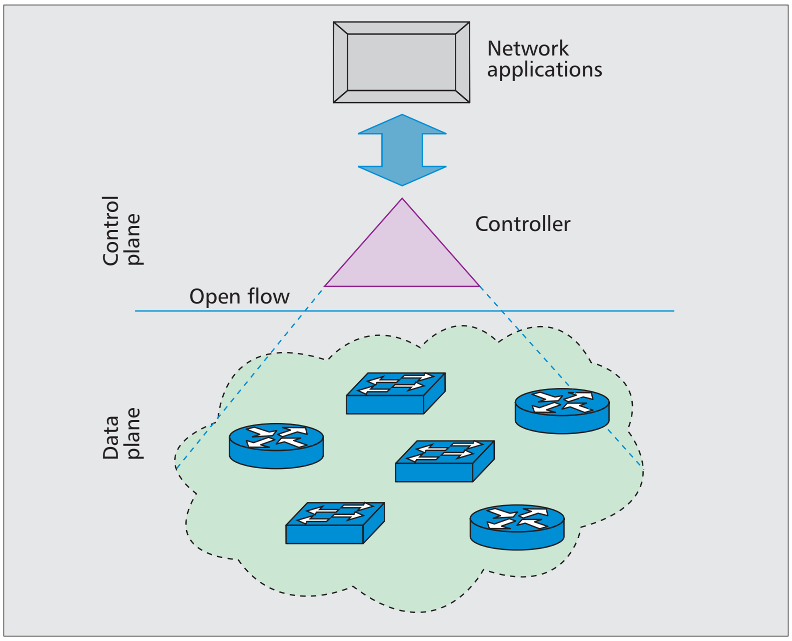

Figure 1. SDN architecture.

- SDN proposes a logically centralized controller solution, without providing any detail on decentralizing controller functionality.
- Compared to the online processing speed in a traditional router, the flow setup delay introduced by switch to controller communication can be significantly high in WANs.
- SDN has successfully defined the interface for switch to controller communication, but there is no standard interface for controller to application communication. As a result, a number of incompatible controllers (e.g. NOX [10] and Floodlight [11]) have been implemented. This is strangling SDN's goal for neutrality and independent innovation.
- It is hard to achieve data plane security in SDN. Once a flow is established, each packet with the same flow signature is treated as authentic and no inspection is done to identify packet injection attacks. These shortcomings have to be addressed to ensure a widespread acceptance of the SDN technology.

Design Considerations

There exist a number of alternatives for designing a single administrative domain (e.g. an ISP or an enterprise network) WA-SDN architecture. Here we discuss some of these alternatives along with their relative advantages and disadvantages.

Single vs. Multiple Controllers

The controller is the most important artifact in an SDN architecture. A single controller solution may result in a single point of failure and performance bottleneck problems in a WA-SDN. The entire network will collapse if the controller fails. On the other hand, no matter where we place the controller it will be farther away from some switches. These switches will experience higher flow setup latency. Clearly a single controller solution is not suitable for WA-SDN.

For a multi-controller WA-SDN solution two

Network management should be simpler and less error-prone in SDN. In contrast to controlling and configuring each switch independently, a network administrator can program the controller, which in turn can configure the switches as needed.

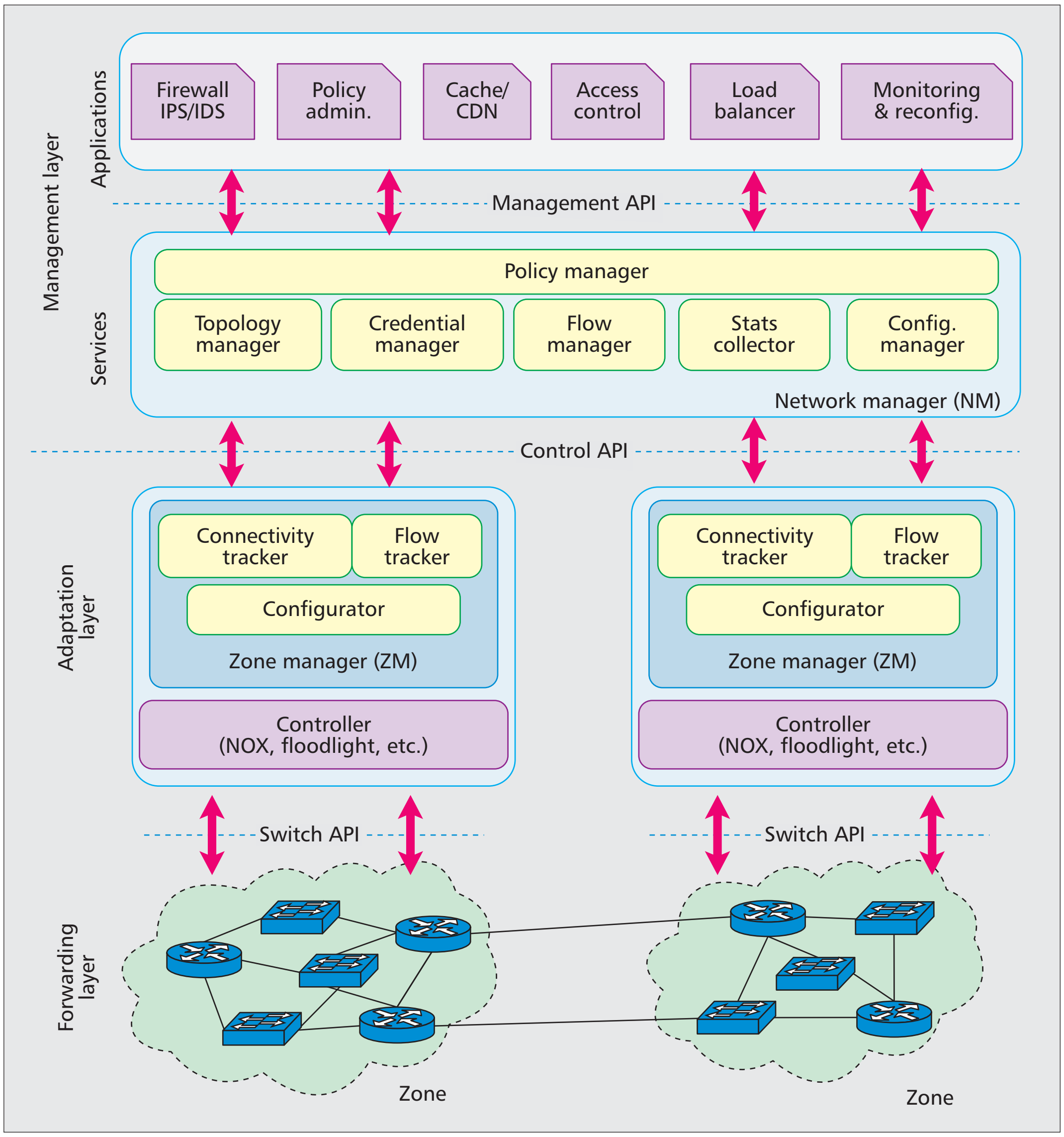

Figure 2. Proposed system architecture.

alternatives are possible: replicated and distributed. Multiple *replicated controllers* can improve fault resilience. A naive approach is to maintain an online shadow controller that will takeover only if the primary controller fails [12]. Each switch is configured to simultaneously communicate with both controllers. This may generate significant communication overhead in a WAN, where the controller and switches may be many hops apart. For short flows this may generate more flow setup traffic than the flow itself. On the other hand in a *distributed controller* architecture [5–7] each controller is responsible for a portion of the network. In general, a distributed controller solution should yield better performance and robustness than a replicated one.

Multi-Controller Architectures: Horizontal vs. Vertical

In a multi-controller setup controllers can be arranged in two ways: horizontal or vertical. In a horizontal setup each controller has the same responsibilities but a partial view of the network. This architecture is proposed in Hyper Flow [7] and Onix [6]. HyperFlow handles state distribution of the distributed controllers through a publish/subscribe system based on the WheelFS distributed file system. Controller state distribution in Onix is managed through a distributed hash table.

On the other hand, in a *vertical setup* controllers are hierarchically arranged, and they have different responsibilities. For example, Kandoo [5] places the controllers in a two-level hierarchy comprising a master controller and multiple local controllers. Local controllers respond to the events that do not depend on global network state (e.g. elephant flow detection), while the root controller handles the events that require global network view (e.g. rerouting elephant flows).

These alternatives have relative advantages and disadvantages. Both of these alternatives can improve switch to controller latency over a single controller solution. Horizontal arrangement can provide better resilience to failure but managing the controllers becomes harder. On the contrary, a vertical arrangement offers simpler network management through the upper level controller, which remains a single point of failure.

Inter-Controller Communication: In-band vs. Out-of-band

In a multi-controller SDN deployment the mode of inter-controller communication can greatly affect the overall system performance. There are two possibilities to choose from: in-band or out-of-band signaling. With *in-band signaling*, controllers use the data plane for connecting to each other. This incurs no additional cost for inter-controller communication. Usually controllers have to synchronize in real-time for quick flow setup and response to network dynamics. In-band signaling may fail to deliver performance guarantee. On the contrary, *out-of-band signaling* requires a dedicated network between the controllers, which incurs additional cost, but it can provide much better response time and does not suffer from potential congestions. Out-of-band communication should be the preferred choice for a high performance WAN deployment.

Flow Setup Model

Different flow setup models are possible in a multi-controller architecture. Here we discuss three possibilities.

1) Master controller: In a vertical setup a local controller can propagate a Packet_In message to the master controller, which can compute the flow-path and can instruct appropriate local controllers to install required flow table entries. This approach can generate optimized flow paths, but the master controller will become a performance bottleneck.

2) Ingress controller: In a horizontal setup the first controller intercepting a Packet_In message can request other controllers along a desired path to install flow table entries in the switches under their control. This approach does not have a performance hot spot, but controllers have to know about each other and the network topology. Master controller and ingress controller approaches are more appropriate for out-of-band signaling.

3) On-Route: In a horizontal setup a controller can independently schedule a segment of an end-to-end flow in the switches under its control based on its local knowledge. This approach is suitable for in-band signaling. It can offer better performance over the previous two, but the quality of flow paths depends on the level of local knowledge at each controller. The On-Route approach can offer a balance between performance and path quality (see the *Flow Setup* section for details).

An Architecture for WA-SDN

In light of the above discussion, a multi-controller architecture in a vertical setup should be the most suitable solution for WA-SDN. The architecture should be independent of the inter-controller communication models. This would allow a service provider to choose between cost and performance. The resulting architecture is presented in Fig. 2. It has two major components: a network manager (NM) and a zone manager (ZM). There are three logical layers: the forwarding layer, the adaptation layer, and the management layer. This section describes the architectural components, while the next section presents a functional overview.

Forwarding Layer and Zones

From a functional point of view, this layer is similar to the data plane in Fig. 1. Contrary to the single-controller scenario, the switching network is logically partitioned into *zones* based on network proximity (for performance reasons) and/or the switch-to-controller communication protocol (for deployment flexibility). Each zone is controlled by a native controller (e.g. NOX, FloodLight etc.). However, end-to-end network operations are transparent to the zones. Two benefits can be harnessed by this partitioning:

Scalability: First, the load on each controller will be reduced, compared to a single controller scenario. This will allow the controllers to maintain fine-grain contact with the switches and perform more accurate monitoring. Second, in a dispersed WAN, a controller can be placed in each geographic location, which will reduce flow setup delay.

Extendibility: The controller-to-switch communication is abstracted by the switch API. This will allow different controller implementations and switch APIs to coexist in the same network. It will facilitate gradual deployment of new or upgraded controller implementations and switch APIs.

The controller-to-switch communication is abstracted by the switch API. This will allow different controller implementations and switch APIs to coexist in the same network. It will facilitate gradual deployment of new or upgraded controller implementations and switch APIs.

Adaptation Layer and Switch API

The purpose of this layer is to leverage the performance and extendibility offered by zone level partitioning. There exist many controller implementations, each having its own advantages and drawbacks, and each offering a different interface (aka North Bound API) to the management applications. Moreover, OpenFlow is not the only *switch API* (aka South Bound API) available. Vendors have their proprietary extensions over the OpenFlow protocol.

ZMs can make the architecture independent of the switch API and controller implementation. A ZM is installed as a plug-in to each controller, which communicates with the switches in its zone using native switch API (e.g. OpenFlow). The ZM, on the other hand, gets commands from the NM using a standard Control API, and translates and feeds those commands to the controller. The *connectivity tracker* and *flow tracker* components in the ZM collect topology and flow information from the controller, respectively, and forward this information to the NM periodically. This setup should not incur significant overhead because the switch to controller traffic is restricted within a zone, and all components within a zone reside in close network proximity. The *Configurator* module receives switch configurations from the NM and translates them into appropriate operations performed on the controller.

Management Services and Control API

The NM should provide the essential management services and a global (abstract) view of the network for management applications. The NM should have the following components:

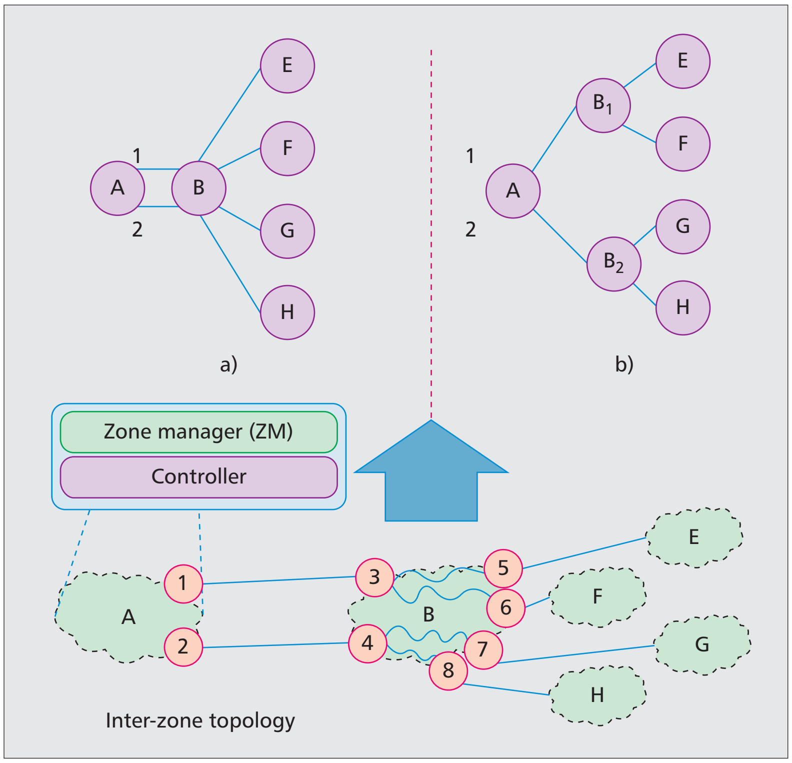

Figure 3. ANT graph example.

Topology Manager is responsible for intercepting topology related updates from the controllers. This module will combine local topology information to form a global topology. This module also creates aggregate topology information and pushes it down to the ZMs to facilitate inter-zone flow setup (further detail follows).

Credential Manager maintains the public keys for the controllers and network applications. As a result, only authenticated controllers can provide information in the system and trusted applications can use this information. The switch API (e.g. OpenFlow) should ensure security between a controller and its switches.

Flow Manager installs and maintains predefined and long lived flows according to the applications and management requirements. This module focuses on ensuring globally optimal flow paths. This module can also be used to shape flow paths and flow volume for adapting to network dynamics.

Stats Collector keeps track of the detail and aggregate flow and topology information to aid monitoring applications. Other modules within the NM can also communicate with the stats collector to obtain network statistics. A hybrid combination of the poll and push strategies for stats collection can produce highly accurate stats with minimal monitoring overhead (explained in the next section).

Configuration Manager maintains up-to-date information about the switches, controllers, and ZMs in each zone. It provides this configuration information to management applications and implements configuration change requests through the configurator modules in the ZMs.

Policy Manager interprets the management API messages. It has the necessary intelligence to interpret global policies and to translate applications' requests to appropriate function calls to the other modules in NM. Existing solutions for policy definition, conflict resolution, and policy based management can be adapted for extending the functionality of this module.

Control API provides the standard communication interface between ZMs and the NM. This allows different controller implementations to coexist. For each controller implementation a different ZM has to be implemented for translating Control API messages to the native controller messages.

Management Applications and Management API

A wide variety of management applications, like firewall, intrusion prevention/detection systems (IPS/IDS), policy administration tools, in-network cache, CDN, user access control, load balancing, monitoring, and reconfiguration, can thrive in this architecture. There are two ways to deploy these applications: as a plugin within the NM or as a standalone component. The second option is more attractive because it provides loose coupling and provision for a wider range of applications. For the second option a RESTful Management API is more desirable, since REST offers benefits like efficiency, scalability, simplicity, and extendibility. This API should expose the functions offered by the management services, like access to up-to-date topology information, manipulate data flows, network statistics at different granularities, configuring network elements, and so on.

Functional Overview

In this section we highlight three important functionalities and how they can be implemented in a WA-SDN architecture.

Distributing Topology Information

Gathering real-time topology information is a difficult task in a WAN. If a central entity (e.g. a single controller) is assigned the task of tracking all the devices and links, it will be hard to gather accurate network topology in a timely manner. One ZM is assigned the responsibility of tracking the devices and links in a specific zone. Since the number of switches and links in a zone can be kept small, a ZM can poll them (via controller) more frequently and can provide timely and accurate topology information.

Each ZM has a detailed topology of its own zone. However, a ZM requires some knowledge about the overall network topology for flow scheduling, load-balancing, and traffic shaping. The NM can collect detailed topology information from each ZM and compute aggregate network topologies (ANT). An ANT is a graph where nodes represent zones and edges represent inter-zone links. The NM can compute and send an ANT for each ZM.

Now we provide a simple method for constructing the ANT graph. First consider a simple case where any pair of zones is connected with at most one link. In this case the ANT graph for any ZM will be the minimum spanning tree rooted at that ZM. However, it is more likely that multiple links will exist between a pair of zones, since zones are logical groupings of switches within a single administrative domain. If we allow multiple edges between a pair of nodes, we cannot ensure an optimal routing path based on an ANT graph. We can explain this situation using the example in Fig. 3. Consider the inter-zone topology in this figure. We want to compute the ANT graph for zone A's ZM. There are two links between zones A and B. From zone A's perspective, nodes 3 and 4 are ingress and nodes 5 to 8 are egress. Assume that a packet can reach egress nodes 5 and 6 in shortest paths if it enters zone B through node 3, while the shortest paths to egress nodes 7 and 8 are through ingress node 4. Now if we allow multiple edges between zones A and B we will obtain the ANT graph (a) in Fig. 3. Consider that zone A wants to set up a flow to zone G via B. Based on this graph, zone A will not be able to pick between links 1 and 2. Now consider the ANT graph (b) in Fig. 3. In this graph zone B is split into B1 and B2, corresponding to ingress nodes 3 and 4. Based on this graph, zone A can readily pick link 2 when it wants to set up a flow to zone G.

Flow Setup

Flow setup is the most frequently executed task by a controller. It will not be scalable if ZMs need to contact the NM to set up every flow. On the other hand, maintaining detailed network topology at every ZM is not scalable due to synchronization overhead. To balance between flow setup delay and synchronization overhead, we chose the On-Route flow setup model. The NM can compute and store the ANT graphs in each ZM. This will enable a ZM to install flow table entries within its own zone without contacting NM and yet can ensure near-optimal inter-zone routing paths.

Figure 4 presents the sequence of operations that takes place to set up a flow within a zone. When a new flow request arrives at a zone's ingress switch (setp 1), the switch contacts its controller for flow setup (setp 2). The controller in turn queries the ZM, which resides in the same machine (step 3). Now the ZM consults its ANT graph (step 4) to find the egress switch (node d in this example) along the global optimal path to the target zone. Then the ZM instructs the controller (step 5) to initiate a flow from the ingress switch (here a) to the egress switch (here d) (steps 6 and 7). Finally, the flow traverses the zone (step 8).

By repeating this process at each intermediate zone, an end-to-end flow can be established. A ZM can ensure shortest path routing in the ANT graph and within its own zone. Hence, according to the optimality principle the path from source node to the ingress switch of the destination zone will be the shortest. However, the last portion of the path at the target zone may not be the shortest for a specific case. Suppose the target zone is Z and the flow enters zone Z from zone Y. If there are multiple links between zones Y and Z, and the flow terminates at an internal switch (i.e. other than the border switches) in zone Z, then the ZM in zone Y will have no way from the ANT graph about which

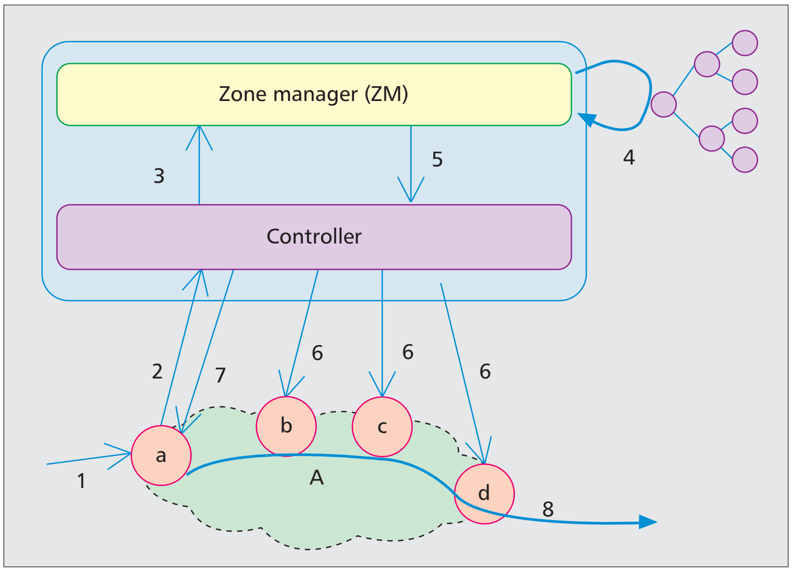

Figure 4. Intra-zone routing.

link to choose. This is because the ANT graph does not have information about the internal switches in a zone and zone Y cannot determine which ingress switch in zone Z has a shorter path to the target switch. However, this special case will introduce only a minor deviation from the optimal routing path.

Monitoring

Monitoring is essential to track and ensure proper functioning of a network. As discussed earlier, there are two modes of monitoring: push and poll. Push is more bandwidth efficient, as in this case the monitored entity updates the monitoring entity when some event occurs. Implementing a push service within a switch is against SDN principle because it will increase a switch's complexity. The other alternative is to poll the monitored device periodically. Polling frequency determines the accuracy of collected data. But increasing polling frequency incurs additional network overhead. For this reaso- most of the switch APIs support poll based monitoring.

Considering these trade-offs, we can adopt a hybrid strategy. As depicted in Fig. 5 a monitoring application registers itself with NM using the management API. The application provides its requirements and desired update frequency to the NM. The stats collector module in NM contacts and registers with appropriate ZMs depending on the application's requirements. Accordingly, the ZMs start polling the controllers, which in turn collect the desired statistics from the switching fabric and updates the ZMs. The ZMs accumulate and aggregate raw data from the switches and send them to the NM, which in turn updates the application. A similar approach for monitoring in single-controller SDN can be found in [13].

This hybrid strategy can achieve a higher polling frequency than a central controller scenario, because there are multiple ZMs/controllers in the system, each polling a small number of switches. Moreover, the traffic between ZMs and the NM will be low as the ZMs push aggregated

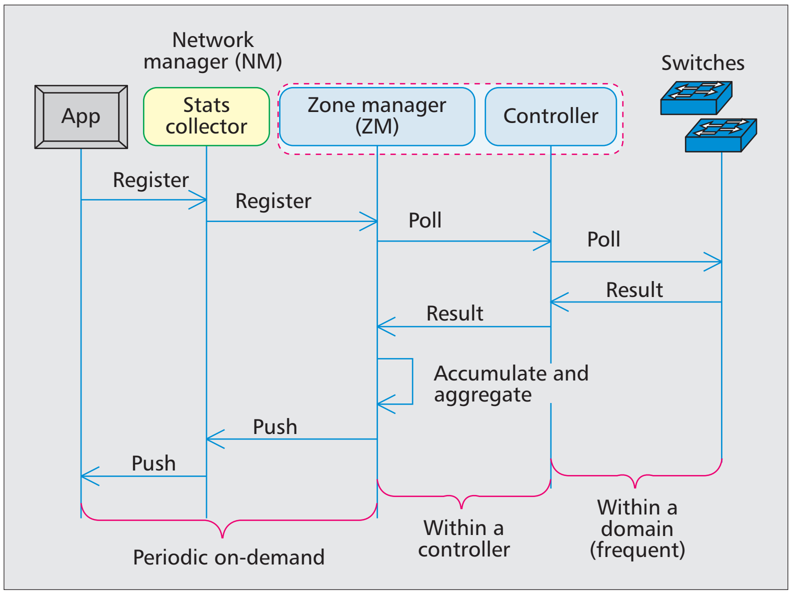

Figure 5. Monitoring process.

information to the NM on demand only. This will result in a lower load at the NM as well. Hence, better accuracy can be achieved at the expense of a lower monitoring traffic.

Related Architectures

In this section we compare the discussed architecture with existing multi-controller SDN architectures that target enterprise networks. In particular we focus on four prominent multi-controller SDN architectures: Google's *B*4 network [8], Kandoo [5], HyperFlow [7], and Onix [6].

Google's inter-datacenter network B4 is the largest SDN deployment in the industry. B4 network provides the backbone connectivity between 12 sites (datacenters) from Google that span the Globe. Each site is controlled by a set of OpenFlow controllers, which directly connect to the switches in datacenters. The controller from each site communicates with a central component called the SDN gateway, which is analogous to the NM. The SDN gateway is mainly used for global traffic engineering and better fault resilience. The controllers communicate with each other (i.e. horizontal setup) to exchange topology and flow information, which adds a lot of complexity and overhead in the B4 architecture.

Kandoo [5] proposes a two-level controller network (i.e. vertical setup) consisting of a root controller (analogous to NM) and multiple local controllers. The root controller has a global knowledge of the network, while the local controllers know only about their own zone. Local controllers report to the root controller. Local controllers try to handle OpenFlow events using their local knowledge. A local controller contacts the root controller if it finds its local knowledge inadequate to handle a packet. As a result, the root controller has to process a significant number of events. On the contrary, ZMs can handle flow setup requests independently using detailed local topology and the ANT graph. NM is contacted only to update the network topology and flow statistics. This should provide a better separation of concerns, allow lower flow setup delay, and lesser load at the NM. Results from our work on controller placement in multi-controller SDN can well support this conjecture (Fig. 4 in [14]).

Both Onix [6] and HyperFlow [7] propose a fully distributed controller network without any central component (i.e. horizontal setup). Onix maintains a distributed data structure called Network Information Base (NIB) for synchronizing controllers' states and for accessing devices. NIB is a graph representing the network topology. Each device has a set of key-value pairs in NIB. Topology information (mostly static) is fully replicated across all controllers, whereas dynamic state information (e.g. link utilization) is indexed in a Distributed Hash Table. Each controller can aggregate the network topology it controls and expose it as a single node to other controllers. HyperFlow, on the other hand, relies on the publish/subscribe mechanism of the WeelFS [15] file system for synchronizing controller states. Both of these approaches provide a logically centralized view of the network to a network application. However, in both of these architectures, applications are responsible for handling stale data and inconsistent network views. This increases the complexity of a network application.

Conclusion

A complete management framework is essential to realize SDN's promise for a simple, manageable, and cost effective networking solution. In this work we discussed the design rationales and the architectural alternatives for WA-SDN solutions. We have provided an overview of a desirable architecture to fulfill these requirements and have explained the major functionalities in this architecture. We are gradually elaborating and implementing the components and mechanisms in this architecture. We are maintaining our related research outcome (in terms of publications and downloadable software components) in our project page at www.waterloosdn.org. We believe that the design rationales and architectural alternative presented in this article will serve as a cornerstone for future WA-SDN management architectures.

References

[1] N. McKeown *et al.*, "Openflow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, 2008, pp. 69–74.
[2] M. Casado *et al.*, "Ethane: Taking Control of the Enterprise," *Proc. 2007 Conference On Applications, Technologies, Architectures, and Protocols For Computer Commun.*, 2007, pp. 1–12.
[3] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," *IEEE Commun. Mag.*, vol. 51, no. 2, 2013, pp. 114–19.
[4] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, 2013, pp. 136–41.
[5] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," *Proc. HotSDN 2012*, pp. 19–24.
[6] T. Koponen *et al.*, "Onix: A Distributed Control Platform for Large-Scale Production Networks," *Proc. OSDI 2010*, pp. 1–6.

[7] A. Tootoonchian and Y. Ganjali, "Hyperflow: A Distributed Control Plane for Openflow," *Proc. 2010 Internet Network Management Conf. Research on Enterprise Networking*, ser. INM/WREN'10, Berkeley, CA, USA: USENIX Association, 2010, pp. 3-3.

[8] S. Jain *et al.*, "B4: Experience with a Globally-Deployed Software Defined WAN," *Proc. ACM SIGCOMM 2013 Conf.*, SIGCOMM, ACM, 2013, pp. 3–14.

[9] J.-P. Martin-Flatin, S. Znaty, and J.-P. Hubaux, "A Survey of Distributed Enterprise Network and Systems Management Paradigms," vol. 7, no. 1, March 1999, pp. 9–26.

[10] "Nox," http://www.noxrepo.org/nox/about-nox/.

[11] "Project Floodlight," http://www.projectfloodlight.org/projects/.

[12] H. Kim *et al.*, "Coronet: Fault Tolerance for Software Defined Networks," *Proc. IEEE Int'l Conf. Network Protocols (ICNP)*, 2012.

[13] S. R. Chowdhury *et al*., "PayLess: A Low Cost Network Monitoring Framework for Software Defined Networks," *Proc. IEEE/IFIP Network Operations and Management Symp. (NOMS)*, Krakow (Poland), May 2014.

[14] M. F. Bari *et al.*, "Dynamic Controller Provisioning in Software Defined Networks," *Proc. IEEE/ACM/IFIP Int'l Conf. Network and Service Management (CNSM)*, Zurich, Switzerland, October 2013.

[15] J. Stribling *et al*., "Flexible, Wide-Area Storage for Distributed Systems with Wheelfs," *Proc. 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI '09)*, Boston, MA, April 2009.

Biographies

REAZ AHMED is working as Associate Professor in the department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh. He received the Ph.D. Degree in Computer Science from the University of Waterloo in 2007. He received the MSc. and BSc. degrees in Computer Science from BUET in 2002 and 2000, respectively. He received the IEEE Fred W. Ellersick award 2008. His research interests include future Internet architectures, wide area service discovery and content sharing peer-to-peer networks with a focus on search flexibility, efficiency, and robustness.

RAOUF BOUTABA [F] received the M.Sc. and Ph.D. degrees in computer science from the University Pierre & Marie Curie, Paris, in 1990 and 1994, respectively. He is currently a professor of computer science at the University of Waterloo and a distinguished visiting professor at the division of IT convergence engineering at POSTECH. His research interests include network, resource and service management in wired and wireless networks. He is the founding editor in chief of the *IEEE Transactions on Network and Service Management* (2007-2010) and on the editorial boards of other journals. He has received several best paper awards and other recognitions such as the Premier's Research Excellence Award, the IEEE Hal Sobol Award in 2007, the Fred W. Ellersick Prize in 2008, the Joe LociCero and the Dan Stokesbury awards in 2009, and the Salah Aidarous Award in 2012.