2015 44th International Conference on Parallel Processing

Zebra: An East-West Control Framework For SDN Controllers

Haisheng Yu*, Keqiu Li*, Heng Qi*, Wenxin Li* and Xiaoyi Tao*
*Dalian University of Technology
Email: likeqiu@gmail.com

***Abstract*—Traditional networks are surprisingly fragile and difficult to manage. Software Defined Networking (SDN) gained significant attention from both academia and industry, as if simplify network management through centralized configuration. Existing work primarily focuses on networks of limited scope such as data-centers and enterprises, which makes the development of SDN hindered when it comes to large-scale network environments. One way of enabling communication between data-centers, enterprises and ISPs in a large-scale network is to establish a standard communication mechanism between these entities.**

In this paper, we propose Zebra, a framework for enabling communication between different SDN domains. Zebra has two modules: Heterogeneous Controller Management (HCM) module and Domain Relationships Management (DRM) module. HCM collects network information from a group of controllers with no interconnection and generate a domain-wide network view. DRM collects network information from other domains to generate a global-wide network view. Moreover, HCM supports different SDN controllers, such as floodlight, maestro and so on. To test this framework, we develop a prototype system, and give some experimental results.

***Keywords*-Software-defined networking, OpenFlow, Zebra**

I. Introduction

Although traditional networking has been widely successful and become a globe-spanning infrastructure upon which national economies depend, it has long been known that the traditional network architecture has significant deficiencies, such as the coupling between architecture and infrastructure, low-level configuration of individual components, distributed and specialized middle-boxes, and limited visualized tools. The above problems are not superficial symptoms but are inherent to traditional networks and deep-rooted in the architecture.

The appearance of SDN has given us a new way to evolve the network. SDN advocates the separation of the control plane from the data plane, and aggregates decision-making of the higher-level routing decisions into control layer in SDN, unlike traditional network switch implementations, which is divorced from the data handling layer. A SDN controller's processing capability is limited: NOX [1] could process about 30K requests per second; Floodlight could process about 250K requests per second and Maestro [2] could process about 300K requests per second. To fulfill large-scale network environments and achieve a scalable control layer, many recent papers have explored architectures for building large-scale or global-wide SDN controller [3], [4], [5], and they focus on necessary components to implement such a SDN controller. One key limitation of those systems is that each SDN controller has its own communication mechanism that makes SDN networks very difficult to exchange information between different domains. The domain can be an enterprise network or a data-center, and it can also be an Autonomouse System.

SDN usually has three different function layers, such as application layer, control layer, and data layer. Openflow [6] is a standardized protocol between control layer and data layer, which makes SDN develop quickly. However, there is no standardized protocol within control layer, which makes the development of SDN hampered when it comes to large-scale network environments. Southbound communication refers to network communications between SDN controllers and forwarding devices. Northbound communication refers to network communications between SDN controllers and network applications. East-west communication refers to network communications between SDN controllers.

In this paper, we present a promising SDN architecture Zebra for improving the SDN management. The core component in Zebra is SDN controller decision layer that has two sub-modules, Heterogeneous Controller Management (HCM) and Domain Relationships Management (DRM). To prove the solution, we design and implement a prototype system that use floodlight, ryu and pox as SDN controllers.

The goal of this paper is to resolve communication problems between different controllers. In fact, our research is still at an primary stage and there are many unanswered questions about the architecture. Rather by presenting a specific design alternative that is radically different from already existing controller architecture, our aim is to highlight the importance of general communication framework between SDN domains. We also hope this work will help to focus the standardization of protocols in East-West Control Plane for SDN Controllers.

This paper makes the following contributions:

1. We propose a new layer between application and controller and design a distributed control framework, which makes heterogeneous controller work together.
2. We have designed an initial Zebra architecture, which is capable of making different kinds of controllers constitute a large control plane.
3. To verify Zebra, we design and implement a prototype system. In our prototype, there are three different SDN

0190-3918/15 $31.00 © 2015 IEEE
DOI 10.1109/ICPP.2015.70

controllers: floodlight, pox and maestro. Our experimental evaluation clearly indicates Zebra has higher performance than floodlight and maestro.

The rest of the paper is organized as follows: We describe the background and motivation in Section 2. We present the design and implementation in Section 3 and evaluate the performance of Zebra in Section 4. Finally, we present the related work in Section 5 and conclude it in Section 6.

II. Background and motivation

The concept of SDN domains was introduced to support the need for partitioning a network control plane among different controllers within an administrative domain. An SDN domain can be defined as the portion of the network being managed by a particular SDN controller.

In this section, we want to highlight the importance of communication protocol between SDN domains. We insist that as long as a large-scale network works well it should satisfy the following requirements:

High Scalability. Since SDN controller can feasibly manage limited number of devices, theres doubting that a reasonably large network should deploy more than one SDN controllers.

Enough Privacy. A carrier may choose to implement different privacy policies in different SDN domains. For instance, an SDN domain may be dedicated to a set of customers who implement their own highly customized privacy policies. Then the networking information in this domain (e.g., network topology) should not be disclosed to an external entity.

Great Generality. If SDN replaces the positions of traditional network, a standard communication protocol is needed to deliver necessary message between domains, such as reachability, topology and bandwidth.

Good Compatibility. To improve the compatibility with traditional network, we should understand the relationship between traditional domains. Then we must create a communication protocol or use existing protocols to control the hybrid network.

It is suggested that a standardized control framework can improve innovation and management of Internet.

III. Design requirements of management layer

In this section, we present the design principles of the management layer for Openflow-based networks and how it is designed in Zebra.

A. General SDN management architecture

As we know SDN has a general management architecture. Figure 1 shows the components of the architecture. We first present the general management architecture for SDN networks and then point out open issues and weaknesses of the architecture. The architecture have three different function layers, such as application layer, control layer, and data layer [7], [8]. The application layer determines how a specific application uses a network. The control layer is responsible for collecting network statistics and distributing flow entries. The data layer handles individual packets based on the state that is output by the control layer, and count the packets. Over the years, plenty of controllers were developed for this architecture: Beacon [9], Floodlight [10], NOX [1], POX [11], Ryu [12], Trema [13], OpenDaylight [14] and more recently ONOS [15]. 3-layer architecture has successfully resolved SDN southbound communication problem between controller layer and data layer. Openflow doesn't tell us how does northbound protocol between applications and controllers work well? One reason is that we treat the subject without ever clarifying this issue that the SDN routing problems needs to be decomposed into intra-domain relationship problems and inter-domain relationship problems.

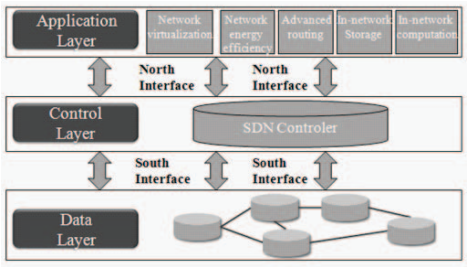

Figure 1. Common SDN has three layers: application layer, controller layer and data plane layer

The fundamental assumption in 3-layer architecture is that the southbound interface between data layer and control layer, and the northbound interface between application layer and control layer are standardized. Actually, the south interface changes all the time. Worse still, there are many types of SDN controllers that cause the south interface impossible to standardize. Each SDN controller has their own interface for application. Although all SDN controllers share a common base interface set, when you move up the ranges, you'll find more and more different advanced features . For instance, both pox and floodlight have interface to get switch and flow information, and it is very easy to delete a flow in floodlight using static-entry-pusher interface with delete action, it is very hard to delete a flow in pox controller. Meanwhile, their functions are also different. For instance, OpenDaylight [16], ryu [12] and floodlight [10] all have interface to get switch and flow information, but a firewall only exists in OpenDaylight, a load balancer in Ryu and a monitoring application in Floodlight.

B. Layering

To address these architectural issues, many efforts have been put to find an effective way to redesign the Internet [17], [18], [19], [20], [21]. 4D and RCP are two well-known researches of them.

The 4D project, starting in 2004, advocated a clean slate design and proposed three key principles: network-level objectives, network-wide views, and direct control. It emphasized separation between the routing decision logic and the protocols governing the interaction among network elements. It proposed giving the "decision" plane a global view of the network, serviced by a "dissemination" and "discovery" plane, for control of a "data" plane for forwarding traffic.

There has been substantial research on how to build SDN management layer [22], [23], [24], [25], [3], [4], [26], [27], but these efforts have primarily focused on networks of limited scope such as data-centers and enterprises. However, Internet access is provided by ISPs that employs a range of technologies to connect users to their network. There are always three tiers in ISP. ISPs requiring no upstream and having only customers (end customers and/or peer ISPs) are called Tier 1 ISPs. Tier 2 ISPs are also known as Transit ISPs. Just as their customers pay them for Internet access, Transit ISPs themselves pay upstream ISPs for Internet access. An upstream ISP usually has a larger network than the contracting ISP or is able to provide the contracting ISP with access to parts of the Internet the contracting ISP has no access to. Tier 3 ISPs provide Internet access for end users. Internet access is the consumer goods in ISP, while it is the flow entries in Zebra. We propose Zebra, it can make a clear separation between application layer and control layer, which makes network administrator much more smoothly manage network. There are also three tiers in Zebra architecture. Data layer is servered as end users in ISP.

Zebra shares the 4D's view that decision function and dissemination and discovery function should be dispersed to multi-layers [17].

C. Northbound problem

There are many successful solutions in resolving SDN southbound communication problem, but very few in resolving northbound problem between application layer and controller layer. The main reason is that we treat the subject without ever clarifying this issue that the SDN routing problems need to be decomposed into intra-domain relationship problems and inter-domain relationship problems.

This paper is the first piece of work to tackle the above-mentioned problems. Zebra can make a clear separation between application layer and control layer; Decision Layer is the core component of Zebra, which acts as a standard communication protocol between application layer and controller layer; Data layer consists of switches performing flow-based packet forwarding, and dissemination layer consists of one (or possibly more) controller works like cache server. Decision layer centralizes all the network intelligence and network control, such as routing decisions and QoS control. Zebra architecture is depicted in Figure 2. It shows decision layer is a new layer between application layer and controller layer, which manipulates different controller's interface and provides application layer with a standard interface. We also build a prototype to demonstrate the efficacy of Zebra design.

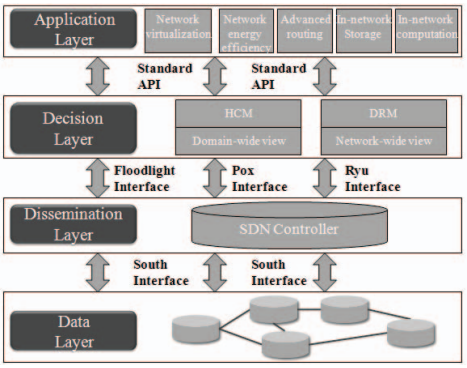

Figure 2. Zebra architecture has four layers, decision layer is a newly established layer between applications and controllers

D. Domain tasks

For a large-scale network, it is not enough to have only a single controller with respect to scalability and reliability. For instance, a SDN controller's processing capability is limited: NOX [1] could process about 30K requests per second; Floodlight could process about 250K requests per second and Maestro [2] could process about 300K requests per second. To fulfill large-scale network environments and achieve a scalable control layer, it is necessary to allow multiple controllers to control multiple domains at the same time.

To address these issue, many early researches were struggling to find an effective way to decompose Internet service into well-defined tasks. RCP [18] collects information about external destinations and internal topology and selects the BGP routes for each routes in an AS. RCP comprises of three modules: the IGP Viewer, the BGP Engine, and the Route Control Server. The IGP viewer establishes IGP adjacencies to one or more routes, which allows the RCP to receive IGP topology information. The BGP engine learns GBP route assignments to each routes. The route Control Server then uses the IGP topology from the IGP Viewer information and the BGP routes from the engine to compute the best BGP route for each route. SDIA [24] breaks down the task of providing connectivity between hosts in different domains

into the following tasks: inter-domain tasks, intra-domain transit tasks and intra-domain delivery tasks.

Zebra decomposes Internet service into interior domain tasks and exterior domain tasks. Interior domain tasks include collecting network information from a group of controllers with no interconnection and generating a domain-wide network view. Exterior domain tasks include collecting network information from other domains and generating a global-wide network view.

E. Network view

Network view is an important basis for the decision of network routing. Many SDN controllers provide a global network view in the management layer. Onix use Network Information Base Details(NIB) to store network view. It holds a collection of network entities, each of which holds a set of key-value pairs and is identified by a flat, 128-bit, global identifier [28].

In the initial SDN architecture, the control plane is mainly realized by a single controller, so the developers can focus on features and functionality, as well as performance.

Zebra devides network view into two parts, namely global-wide network view and domain-wide network view. Domain-wide network view only stores information in a domain, whereas global-wide network view stores a domain's connection with other domains.

IV. Design and implementation

Zebra descends from a long line of work [17], [28], [18], [24], [29]. Zebra divides control plane into Dissemination layer and Decision layer.

A. Zebra architecture

Understanding how Zebra realizes a control platform requires knowing the function of each layer. There are four layers in a network controlled by Zebra, and they have very distinct roles (see Figure 2). Compared to 3-layer platform, we represent significant disparities and major challenges in designing Zebra.

Application Layer: customized demand. The application layer determines how a specific application use a network. There are network virtualization, network energy efficiency, advanced routing, in-network storage, in-network computation in such applications. A graphical user interface (GUI) is needed for SDN controller, then the GUI will change SDN control platform from a software to a networking operating system.

Decision Layer: control logic. Decision Layer is the core component of Zebra, which consists of HCM and DRM. HCM is responsible for routing decisions inside a domain, and CRM is responsible for routing decisions between various domains. Compared to 3-layer architecture, decision layer in Zebra is a newly established layer between SDN applications and controllers. Decision layer makes all decisions driving network control, including reachability, load balancing, access control, security, and interface configuration. Replacing today's management layer, decision layer operates in real time on a network wide view of the topology, the traffic, and the capabilities and resource limitations of the routers/switches. The decision layer uses algorithms to turn network-level objectives (e.g., reachability matrix, load-balancing goals, and survivability requirements) directly into the packet-handling state that must be configured into the data layer (e.g., forwarding table entries, packet filters, queuing parameters). The decision layer consists of multiple servers called decision elements that connect directly to the network.

Dissemination and Discovery Layer: cache server. Dissemination Layer in Zebra acts as an intermediary between potential hundreds of switches and remote application servers by congregating requests from switches into various servers. In the process, a SDN controller frequently requests resources to avoid contacting the server repeatedly for the resource and the route information (flow table) has not changed. The dissemination Layer provides a robust and efficient communication substrate that connects routers/-switches with decision elements. While control information may traverse the same set of physical links as the data packets, the dissemination paths are maintained separately from the data paths. So they can be operational without requiring configuration or successful establishment of paths in the data layer. In contrast, in today's networks, control and management data are carried over the data paths, which need to be established by routing protocols before use. The dissemination layer moves management information created by the decision layer to the data layer and sends state identified by the discovery layer to the decision layer, but does not create state itself.

Data Layer: forwarding data. The data layer is the carrier of SDN network, includes network switches, and any other network elements that support an interface allowing Dissemination Layer to read and write the state controlling the element's behavior (such as forwarding table entries). The function of the data layer is mainly confined to packet forwarding and simple processing. However, it is necessary to build a flexible and easily configured SDN data layer in order to adapt changing demands placed on the application by the end users, the new personalized requirements of data center network and other network application scenarios.

B. Heterogeneous Controller Management Layer

Heterogeneous Controller Management(HCM) is responsible for managing different controllers, such as floodlight, pox and maestro. HCM gathers information from these controller's API and other event source systems, such as network monitoring system.

HCM only considers the flow entries export policies in a domain, while policies between domains is resolved by

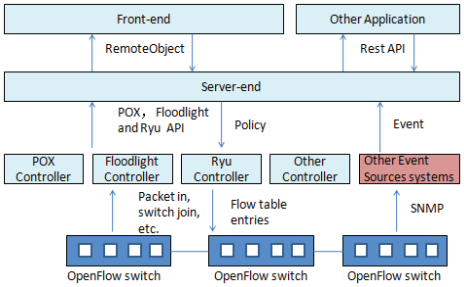

Figure 3. There are five components in HCM, such as frontend, server-end, control plane, data layer and database.

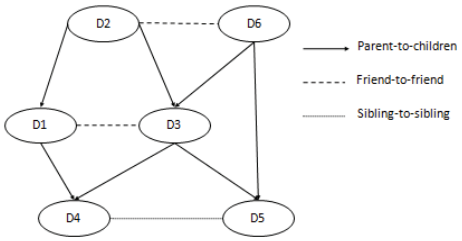

Figure 4. Route paths (D1,D2,D3) and (D1,D2,D6,D3) are valley-free while Route paths (D1,D4,D3) and (D1,D4,D5,D3) are not valley-free.

Domain Relationships Management(DRM).

Figure 3 shows that there are five components in HCM, such as front-end, server-end, control plane, data layer and data base. The front-end is responsible for displaying and managing flow entry, traffic and topology information in SDN. Server-end aims at collecting network topology information and statistics, and updating flow entry according to the change of network topology information or configuration. The control plane only distribute flow entry to switches and send statistics to Server-end. The function of the data layer is mainly confined to packet forwarding and simple processing.

Front-end interacts with server-end by one of java communications flex three ways: flex communicate with ordinary java class RemoteObject. We make use of the Spring DAOs and SQL Maps defined by iBATIS ORM, and connect to the database using Spring JDBC. The SDN controllers provide APIs in the interaction with server-end.

C. Domain Relationships Management Layer

For a large-scale network, it is generally known that different domains should exchange information. Before design a large-scale SDN network, it is very important to define inter-domain relationships. The knowledge of inter-domain relationships has many applications and usage in routing decision. First, it is crucial in network service management decision including the optimal placement of controllers and switches. Second, it can help domain administrators to achieve load balancing, congestion avoidance and fault tolerance. Third, it can aid domain administrators in planning for future contractual agreements. Fourth, it can help domain administrators to reduce the effect of the misconfiguration and to debug switch configurations.

To address this problem, we propose a controller graph representation that classifies domain relationships between children-to-parent (c2p), friend-to-friend (f2f), and sibling-to-sibling (s2s). In the c2p category, a children pays a provider for any traffic sent between themselves. In the f2f category, two domains freely exchange traffic between themselves and their childrens, but do not exchange traffic from or to their providers or other friends. In the s2s category, two domains administratively belong to the same organization and freely exchange traffic between their parents, childrens, friends, or other siblings. Figure 4 gives a example of domain relationships. Our solution is rather standard, borrowing heavily from long standing Autonomous System (AS) deployment practices.

ISPs may friendly exchange information, where multiple ISPs interconnect at Internet exchange points (IXs), allowing routing of data between each network, without charging one another for the data transmitted‡data that would otherwise have passed through a third upstream ISP, incurring charges from the upstream ISP.

Table I
ZEBRA EXPORT POLICY

Route Export	Export Policy
Children to parent	Only routes received from children and sibling
Friend to friend	Only routes received from children and sibling
Parent to children	All routes
Sibling to Sibling	All routes

Table 1 shows common relationships between domains and the export policies associated with them. Parents provide transit to childrens; friends exchange only traffic that is sourced and sinked by them, their childrens or their siblings; and siblings provide mutual transit.

D. Zebra Database

As shown in table 2, there are ten core tables in Zebra database. The table gives Zebra database table names and explanations. Eight of ten tables illustrate the information in a domain: basic information, controller, flow, host, link, switch, switch port and switch port statistics. Meanwhile two of ten tables give information between domains. The s_ip_domain table displays which IP networks is included in a domain. The s_domain_info table shows the relationship between domains as children-to-parent (c2p), friend-to-friend (f2f), or sibling-to-sibling (s2s).

V. EVALUATION

In this section, we evaluate Zebra in two ways: with a test application, designed to test Zebra's performance as a

Table II
TEN CORE TABLES IN ZEBRA

Table	Explanation
s_controller_info	controller type,such as pox,floodlight
s_ip_domain	ip belongs to domain information
s_domain_info	domain information
s_domain_relation	relation between domain
s_flow_info	flow entry information
s_host_info	host information
s_link_info	switch connecting information
s_port_info	switch port information
s_port_stats	port statistics information
s_switch_info	switch information

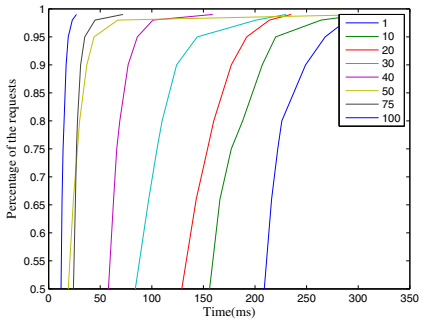

Figure 5. Thread modification percentage of the requests complete in a certain time(ms)

general platform, and with cbench, used to verify DRM's performance. We test two key aspects of a single Zebra instance: request completion time and memory usage, and two key scalability-related aspects of DRM: throughput and latency.

Each HCM instance has to connect the controllers it manages. To stress this interface, we connected three controllers to a single HCM instance and ran apache benchmark to test Zebra's performance as a general platform. The HCM is the focal point of Zebra, and the performance of Zebra will depend on the HCM capacity for processing updates. To measure this throughput, we ran a apache benchmark which repeatedly acquired exclusive access to the HCM. Figure 5 shows that thread modification percentage of the requests complete in a certain time. When there is only one thread, HCM finishes 95% of the requests in 5ms. When the threads increases, HCM uses much less time to complete a request. As you can see from Figure 6, it wastes no more than 3ms to complete 95% of the requests.

Because decision layer in Zebra is a newly established layer between applications and controllers, it is necessary to compare the request time before and after decision layer established. To test the request time, we write an application which sends 100000 flow table entry update requests and monitor the time used in processing the requests. Figure 6 shows that request completion time between Zebra and floodlight is less than 5%. Zebra request's completion time

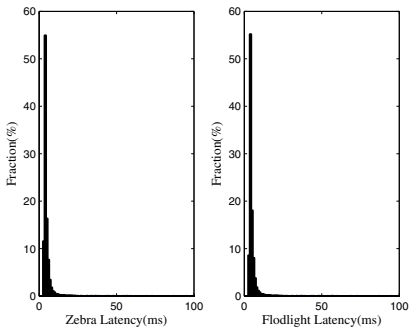

Figure 6. Request completion time compare between Zebra and floodlight

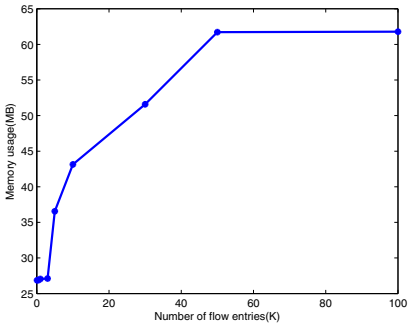

Figure 7. Memory usage to add 1 million flow entries

includes floodlight completion time, so the former spend longer time than the latter. More than 60% of the requests in Zebra is finished in 5ms, and more than 95% of the requests finished in 10ms, so does floodlight. The result of this is identical to that of we do in the previous test. In the worst case, a flow table entry updates requests in Zebra can be finished in 100ms.

Figure 7 describes the memory usage status of Zebra when adding flow entries. It shows that Zebra needs at most 60MB memory to add 1 million flow entries.

Table III
PC DISPOSITION

Product Name	ThinkPad T440
CPU	Intel(R) Core(TM) i5-4200U
Standard Memory	DDR3L-1600 8 GB
Operating System	Windows 7 Professional

In our prototype, each domain has a IP Range or a IP Range set. In our test, we construct two domain: Domain A and Domain B. Domain A's IP Range is 192.168.0.1/24, and Domain B's IP Range is 192.168.1.1/24. We do this test on two PCs with same disposition: one host in Domain A and another host in domain B. Table III summarizes the parameters of the PC. Figure 8 describes the throughput of each controller connected 16 switches changes with the

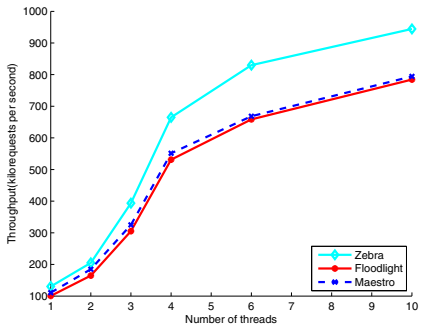

Figure 8. The throughput of each controller connected 16 switches changes with the number of thread

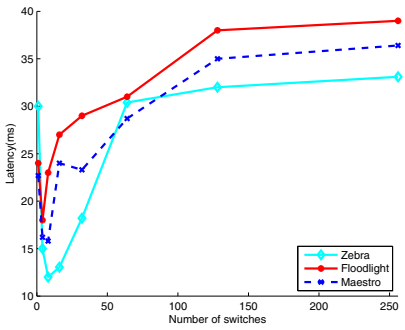

Figure 9. Response time varying the number of switches for runs with 1 threads

number of thread. Floodlight and maestro can process 700 kilorequests per second at most. Zebra DRM can process 900 kilorequests per second. Floodlight and maestro do routing based on network wide view while Zebra do routing based on simplified network wide view.

Figure 9 describes response time (milliseconds) varying the number of switches for runs with 1 threads. At the beginning, adding more CPUs beyond the number of switches improve a little latency, however serving far larger number of switches than available CPUs results in a noticeable increase in the response time.

VI. Related work

In DIFANE [26], "ingress" switches redirect packets to "authority" switches that store all the forwarding rules while ingress switches cache flow table rules for the future use. The controller is responsible for partitioning rules over authority switches.

HyperFlow [5] uses a logically centralized but physically distributed controller in which switches connect to the physically closest part of the controller, which updates the other physical machines on network events via a publish/subscribe system. By passively synchronizing network-wide views of OpenFlow controllers, HyperFlow localizes decision making to individual controllers, thus minimizing the control plane response time to data plane requests. However, if a HyperFlow controller fails, its switches must be reconfigured to connect to a new controller.

ElastiCon [30] addresses controller failure problem by proposing a dynamic migration protocol between controllers, and implements a dynamic load balancing system based on the protocol.

Onix [28] was the first logically centralized but physically distributed controller to implement a global network view. To improve the scaling, Onix supports the creation of new Onix instances with new scopes through aggregation or partitioning. Because the new scope is restricted to devices that are physically close to each other, Onix is a flat architecture which only supports one level of precesses. Furthermore, because it is the applications to resolve the conflict problem when it happens in the process of aggregation or partitioning, which is not easy to define the sub-scope induced by a given policy.

Onix and HyperFlow decreases the look-up overhead by enabling communication with local controllers, while still allowing applications to be written with a simplified central view of the network. The potential downside are trade-offs related to consistency and staleness when distributing state throughout the control plane, which has the potential to cause applications believe that they have an accurate view of the network to act incorrectly.

To get high availability and scale-out, ONOS [15] adopts a distributed architecture and provides a global network view to applications. Zebra uses a hierarchical distributed architecture that differs from ONOS.

FRESCO [31] uses independent modular libraries, and assembles them to provide complex network functions. Our approach, on the other hand, tries to provide complex network functions based on existed controllers. Software-Defined Internet Architecture(SDIA) [24] focuses on how to decompose Internet service into well-defined tasks and how to implement those tasks in a modular fashion. Our approach focus on using Zebra instead of 3-layer architecture to solve communication problems between heterogeneous controllers.

VII. Conclusions

This paper presents the first effort that use SDN four-layer platform to solve multi-SDN controller problems. The proposed solution, Zebra, might provoke interesting discussions on the research community and open the door to a wide range of innovation opportunities. In a word, we expect to see a new generation of SDN that is versatile, flexible, and easy to manage. The proposed architecture has been designed and implemented in the CENI project.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation for Distinguished Young Scholars of China (Grant No. 61225010); the State Key Program of National Natural Science of China(Grant No. 61432002); NSFC Grants 61173161, 61173162 and 61272417; the Fundamental Research Funds for the Central Universities; Prospective Research Project on Future Networks from Jiangsu Future Networks Innovation Institute.

REFERENCES

[1] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, pp. 105–110, 2008.

[2] Z. Cai, *Maestro: Achieving Scalability and Coordination in Centralized Network Control Plane*. PhD thesis, Rice University, 2011.

[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 *Proceedings of the ACM SIGCOMM 2013 conference*, pp. 3–14, 2013.

[4] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013 conference*, pp. 15–26, 2013.

[5] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," pp. 3–3, 2010.

[6] 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 Computer Communication Review*, pp. 69–74, 2008.

[7] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communication Surveys and Tutorials*, 2013.

[8] Y. Wang and I. Matta, "Sdn management layer: Design requirements and future direction," in *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pp. 555–562, 2014.

[9] "Beacon project." https://openflow.stanford.edu/display/Beacon/.

[10] "floodlight project." http://www.projectfloodlight.org/floodlight/.

[11] "pox project." http://www. noxrepo.org/pox/about-pox/.

[12] "ryu project." http://osrg.github.io/ryu/.

[13] "trema project." http://trema.github.io/trema/.

[14] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *2014 IEEE 15th International Symposium on*, pp. 1–6, 2014.

[15] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1–6, 2014.

[16] "Opendaylight project." http://www.OpenDaylight.org/.

[17] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4d approach to network control and management," *ACM SIGCOMM Computer Communication Review*, pp. 41–54, 2005.

[18] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pp. 15–28, 2005.

[19] M. Handley, E. Kohler, A. Ghosh, O. Hodson, and P. Radoslavov, "Designing extensible ip router software," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pp. 189–202, 2005.

[20] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "Sane: A protection architecture for enterprise networks.," in *Usenix Security*, 2006.

[21] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review*, pp. 1–12, 2007.

[22] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: a retrospective on evolving sdn," in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 85–90, 2012.

[23] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent updates for software-defined networks: Change you can believe in!," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, p. 7, 2011.

[24] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-defined internet architecture: Decoupling architecture from infrastructure," 2012.

[25] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, 2009.

[26] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," *ACM SIGCOMM Computer Communication Review*, pp. 351–362, 2011.

[27] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, *et al.*, "Composing software defined networks.," in *NSDI*, pp. 1–13, 2013.

[28] 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 *OSDI*, pp. 1–6, 2010.

[29] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory networking: An api for application control of sdns," in *ACM SIGCOMM Computer Communication Review*, pp. 327–338, 2013.

[30] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. R. Kompella, "Elasticon: an elastic distributed sdn controller," in *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems*, pp. 17–28, 2014.

[31] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks.," in *NDSS*, 2013.