DMC: Distributed Approach in Multi-Domain Controllers

Shahzoob Bilal Chundrigar[1], Min-Zheng Shieh[1], Li-Ping Tung[2], and Bao-Shuh Paul Lin[1,2]
[1]Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
[2]Microelectronics and Information Research Center, National Chiao Tung University, Hsinchu, Taiwan
Shahzoob.bilal@yahoo.com, mzshieh@nctu.edu.tw, lptung@nctu.edu.tw, bplin@mail.nctu.edu.tw

***Abstract*— Multi-domain networks are vital to datacenters, home and enterprise networks. The networks require an independent and private control plane. Thus, the networks must be resilient and easily scalable. The emergence of Software Defined Networking (SDN) protocols simplifies the evolution of networks by decoupling the control plane from the data plane. In this paper, a Distributed approach in the Multi-domain Controllers architecture (DMC) is introduced. It interconnects heterogeneous networks to form a wide area network (WAN) while preserving the privacy of their domains. It also deals with the link failure across the domains, making it resilient. The controller manages its own network domain and exchanges minimal information among neighbor controllers. It applies a light weight control carrier (i.e., RabbitMQ) that reduces overheads. The application has been implemented on top of the RYU control platform.**

Keywords—Multi-domain; Distributed SDN; Multi Controllers;

I. INTRODUCTION

Within the past few years, Software Defined Networking (SDN) [1] has gained immense interest from the industry and academia [2]. It solves many challenging concerns of legacy networks by transferring the intelligence from traditional network devices to a centralized control plane. It has been declared as the future paradigm of networking; aimed at decoupling the control plane from the data plane and network programming. The centralized control plane, often known as the controller, manages the universal view of the network via a southbound interface. OpenFlow [3], specified by the Open Networking Foundation (ONF), is currently the leading protocol specifying the southbound interface.

SDN centralizes the network control plane that provides abilities to program, monitor, and manage networks efficiently. However, a dramatic increase of the network scale may force a single controller to drop an increasing number of incoming packets; it could become the bottleneck of the entire network. This unified approach may not be suitable for interconnecting multi-domain networks while dealing with potential scalability concerns [4].

Recent proposals have been offered to physically distribute the SDN control plane. A distributed control plane is divided into two categories: (1) distributed but logically centralized, and (2) hierarchically distributed.

Hyperflow [5], Onix [6] and Devolved Controllers [7] maintain a network-wide centralized view while distributing the network control plane. In the hierarchical distribution, Kandoo [8] reduces the load of the control plane by processing frequent events in the local controllers and rare events in a global controller. These approaches manage the scalability of the control plane within a single domain. Multi-domain networks are the core of datacenters, home, and enterprise networks. Therefore, a SDN-driven solution is required, where a domain can scale and interconnect amongst other domains resiliently and preserve their privacy.

This paper proposes a novel Distributed approach in Multi-domain Controllers (DMC). It is not only able to handle the scalability of a single domain but can also interconnect multiple domains regardless of the geographical locations of the datacenters, telecommunication, home, and enterprise networks. This paper emphasizes on interconnecting multiple domains but it can be applied to a single domain, intuitively.

In DMC, a centralized controller is in charge of its own domain, and fulfills the necessities of itself and sharing minimal information (i.e., host addresses) in order to entertain end-to-end services among neighbor domains to ensure privacy. Controller to controller communication is made possible by a light-weight control channel that utilizes the messaging mechanism, RabbitMQ [9], which is one of the implementations of AMQP [10]. Additional overheads are reduced because only minimal required information is passed through the light-weight control channel in order to compute routes across domains, as well as to ensure user-to-user connectivity. It manages link failure across the domains and maintains communications smoothly with minimum link recovery time. The DMC is implemented on top of the RYU [11] control platform. The control plane has been declared as the best controller in a recent survey [12]. To the best of our knowledge, this is the first distributed application implemented on the RYU [11] control platform.

The remainder of the paper is organized as follows. In Section II, related work is discussed. Section III introduces the DMC architecture and implementations. Results are presented in Section IV, and Section V concludes the paper.

II. RELATED WORK

The concept of introducing Multi-controllers in the control plane to solve scalability issues is well known, with many

solutions proposed in this area. Hyperflow [5] introduces a cloud of controllers sharing the same consistent network-wide view by synchronization using a distributed file system. Onix [6] takes it one step further and introduces the platform on which a distributed control plane could be implemented. It also uses a distributed file system. ElastiCon [13] and Devolved controllers [7] address the control plane scalability issue in datacenters by dynamically configuring switches to controllers. ASIC [14] introduced the traditional load balancer in their architecture, which balances the load by diverting incoming packets to different controllers.

Kandoo [8] takes the scalability concern to another level by introducing a hierarchical architecture. It proposes two layers of controllers: (1) the root controller maintains network-wide view and processes rare events, whereas (2) the local controllers do not have any knowledge of the network-wide state and are subjected to frequent events. However, all the above mentioned approaches only deal with the scalability concerns in a single domain.

Zerrik et. al. [15] introduced a decentralized hierarchical architecture comprising of direct and parent controllers. The direct controllers serve local requests, and the parent controllers are responsible for inter-domain requests. This approach interconnects the multi-domain. It neither preserves privacy across multi-domain due to parent controllers having network-wide view, nor deals with link failure at the switch granularity level.

DISCO [16] introduced a semi-distributed architecture, where controllers are in charge of their domains and pass the information to other domains via a logical channel. Controllers utilize this information to gain a network-wide view. It uses mainly agents to share the network wide information across the domains. However, it does not preserve privacy as a controller can take over another domain's controller in the case of controller failure.

TABLE I. COMPARISONS OF RELATED WORK

Approach	***Architecture***	***Area***	***Privacy***	***LF***	***CF***
Hyperflow [5]	Logically Centralized	Single Domain	No	No	Yes
Onix [6]	Logically Centralized	Single Domain	No	Yes	Yes
Kandoo [8]	Hierarchical	Single Domain	No	No	No
Zerrik et al. [15]	Hierarchical	(Single, Multi) Domain	No	No	Yes
DISCO [16]	Semi-Distributed	(Single, Multi) Domain	No	Yes	Yes
DMC	Distributed	(Single, Multi) Domain	Yes	Yes	Yes

The related work in terms of issues is summarized in Table I. Five criteria are defined: Architecture, Area (single domain (sd), multi-domain (md)), Privacy (isolation between multiple controllers), Link failure at switch level (LF), and Controller failure (CF). The DMC provides a distributed control plane in the multi-domain networks based upon a messaging mechanism. The DMC scales the control plane that resiliently interconnects multiple domains while preserving privacy by sharing minimal information.

III. ARCHITECTURE AND IMPLEMENTATION

The overall architecture and implementation of the DMC is presented and illustrated in three phases. First, the controller design is mainly based on events and connected via the REST (representational state transfer) interface. Second, routing module, being responsible for computing intra- and inter-domain routes, is described. Finally, the monitoring channel that is subject to any link failure is elaborated.

A. Controller Design

Each domain is supervised by its own independent controller that enables end-to-end services. It establishes a control channel with its neighboring domains to exchange minimal information. This enables end-to-end services without the intervention of a controller in the neighbor's network, resulting in a private domain.

The controller is purely event driven. The overall architecture is depicted in Fig. 1. It provides networking services to its own domain and also communicates with the neighboring domains via the control channel. The channel provides a communication bus in the east/west directions with neighboring domains. It is integrated with the REST interface that allows the user to configure IP addresses on switch interfaces. When data is posted or deleted via the REST

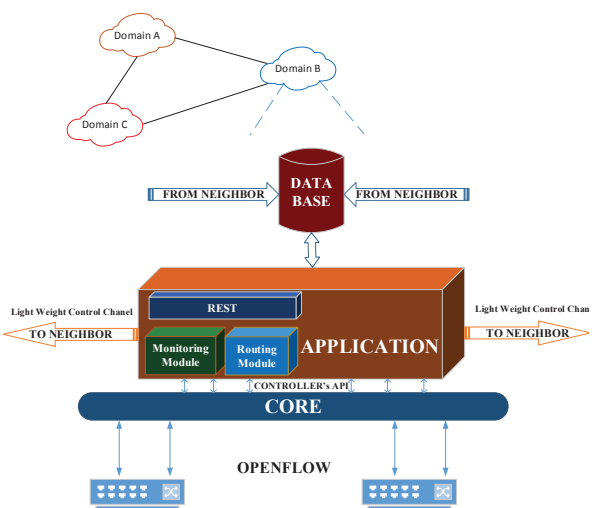

Fig. 1. DMC overall architecture

interface, the controller generates an event that performs three tasks: (1) updates information in the database, (2) updates neighbor domains via the control channel, and (3) requests the controller to perform actions such as setting up the interfaces of a switch.

A centralized database is utilized in each domain to store the information from the neighboring controllers and from its own domain. It stores information such as the categorization of switches (edge and middle) and the network addresses of switch interfaces. The information may be enhanced or used by different modules. The core component provides all the events/API's of OpenFlow [3], enclosed in RYU [11], such as `packet_in`, `packet_out`, etc.

RabbitMQ [9] is utilized to create a light-weight messaging channel. The control channel exchanges minimal information between controllers to carry out end-to-end services. Information that is shared and stored in the database includes the network addresses of switch interfaces in the domain. The Routing Module enriches the information and utilizes it to generate dynamic routes across the domain. The control channel offers a publish/subscribe model among controllers, where each controller is a publisher and a subscriber at the same time.

Both reactive and proactive approaches are used in the architecture in order to keep minimizing the load on controller. All the routes installed by the controller in a switch are prioritized. Proactive flows at the network level are enforced on a switch by applying the REST interface, whereas Reactive flows are calculated dynamically by the controller through the inspection of the first packet. As the first packet is received by the switch, it checks whether the destination network exists in its flow table or not. If it exists, the packet is sent to the controller that floods an ARP (Address Resolution Protocol) request to all the nodes. The destination node responds to the ARP request and hence, the controller knows the MAC address and port connected to that host. Based on that information, the controller installs the high priority flow to the switch.

B. *Routing Module*

The Routing Module is responsible for the calculation of intra-domain and inter-domain routes. Once this module is triggered, it pushes network level routes proactively using the REST interface to the switch, whereas end-to-end routes are calculated actively by the controller. Fig. 2 illustrates a typical implementation of the DMC.

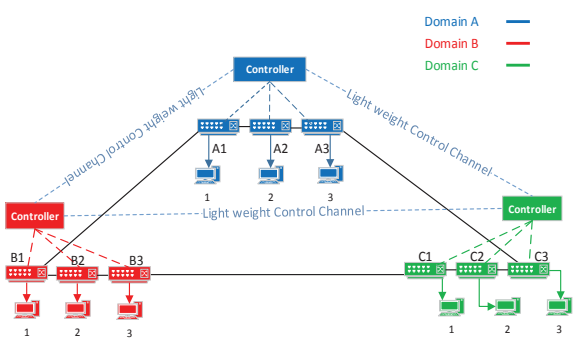

Fig. 2. A typical implementation of DMC

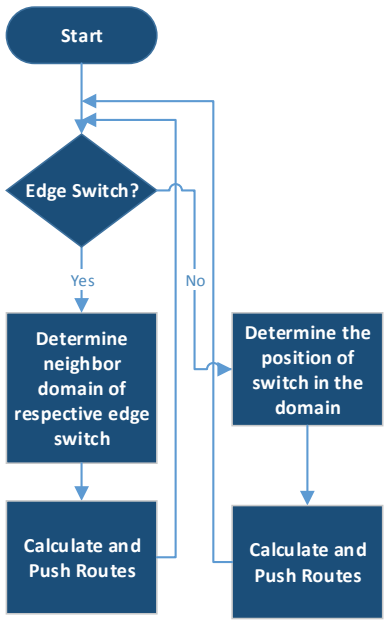

Fig. 3. High level flow chart of routing module

The controller computes the intra-domain routes and sets up gateways for the inter-domain communication. It identifies whether if a switch is an edge or a middle switch. In the case of an edge switch, it locates the connecting domain and sets up the respective edge switch as the gateway. For the middle switch, it finds the position in its domain and sets up its respective gateways.

The high-level flowchart is illustrated in Fig. 3. For better understanding, let's consider the following two scenarios as shown in Fig. 2:

a) Intra-Domain : To develop a route from Switch A1 to A3, since the controller has a global view of its own domain, it recognizes a middle switch between the targets. It identifies the path from A1 to A3 via A2 and vice versa. Thus, the routes are pushed to the respective switches by the controller.

b) Inter-Domain : The objective is to develop a route between Switch A2 and B2 in Fig. 3. Switch A2 is in domain A, and switch B2 is in domain B. In this case, the destination network address is not registered in the controller's domain, and it does not have any view of other domains. It refers to the database and determines which neighbor is registered with the destination network address. After recognizing the domain, it identifies the edge switch connected to that neighbor in Fig. 2. Switch A1 is directly connected to the Domain B, whereas Switch A3 is also connected but it needs to pass Domain C in order to reach its target. The controller picks the best edge connecting switch on the basis of number of hops and identifies the route via Switch A1 to that domain. Consequently, the controller pushes these infomation to the corresponding switch and vice versa.

C. Monitoring Module

It ensures end-to-end connectivity between nodes in the case of a link failure across domains. The goals of a monitoring module are to: (1) trigger an event when a link is compromised, (2) identify an alternative route, (3) delete previous routes, and (4) push new routes via the controller. An OpenFlow event `EventOFPPortsStatus` is used to find the status of the link. The Module is triggered when a link failure occurs, resulting in retrieving the information of the previous route from the database. It deletes the previous route, computes an alternate route, and pushes the route to the switch for further communication.

Consider a scenario in Fig. 2 where Host 2 of Domain A is communicating with Host 1 of Domain B. Here, the routing module has calculated a route where switch A1 and switch B1 are gateways of domain A and domain B, respectively. Once a link failure occurs which results in an triggered event, the monitoring module will be activated and will read the central database and retrieves the previous route. It calculates the second best route on basis of hop counts. The new path is now where switch A3 and switch B3 will serve as the gateways of domain A and B, respectively. Once the routes are calculated, it deletes the previous routes and pushes the new ones via the controller to the switches. It makes the inter-domain connectivity more resilient, resulting in an efficient and reliable network. In the meantime, it guarantees traffic reachability with a fast convergence time.

IV. Evaluation

In order to provide illustrations for how the DMC operates, an emulation experiment was carried out. This is described below.

A. Experiment Setup

In the experiment, three domains enclosed in a private cloud having three hosts each are used. The topology depicted in Fig. 2, represents a WAN covering the three domains. A domain can be any network, ranging from enterprise to home. The DMC is mounted on the RYU [11] control platform. Each centralized controller is in charge of its own network domain and exchanges minimal information with the neighboring domains. The network is emulated by applying Mininet [17]. The Mininet [17] is hosted on a dedicated Virtual Machine (VM), and controllers are hosted on separate VMs. A Single machine is used to host all VMs, mininet and controllers VM uses the LAN segmentation which gives the feel of a WAN. The hosts, connected to the network domains, are user terminals and provided with 100 Mbps links.

1) Flow Setup Time:

The flow setup time is one of the outstanding issues highlighted in [20]. The reactive approach makes an SDN architecture more dynamic and robust but also induces flow setup delays. In the reactive approach, the first packet of the flow is sent to the controller for the route calculation. The DMC offers both approaches to minimize the load on the controller. It also allows the controller to respond to the flows more quickly, making it robust and dynamic.

Considering the example in Fig. 2, proactive flows are installed when the user configures the defined interfaces and the routing module is triggered. These flows are only installed on the network level, concluding that the flows still do not have any node level knowledge. When a host makes a communication path with another host in the same domain or in another domain, the controller pushes the intelligence to the switch at the node level and this makes the communication possible.

The flow setup delay results obtained are presented in Fig. 4, indicating ten different inter-domain flow requests were sent to the controller by different switches. Average response time noticed from the controller is 5.2 milliseconds.

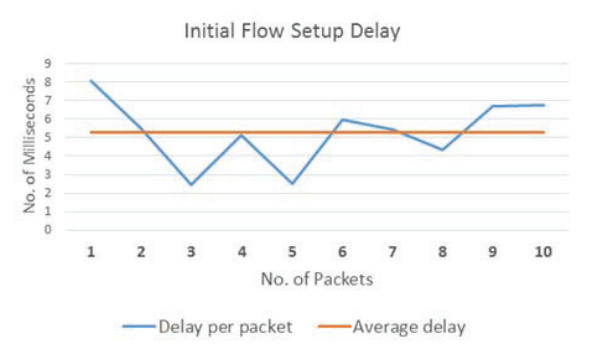

Fig. 4. Flow-setup delay

2) Packet Exchange:

This experiment was conducted to examine the effect of the size of packets exchanged between the domains to make the DMC functional. The DMC utilizes AMQP [10] packets to exchange the minimal information that uses a mere few bytes. The DMC efficiently reduces the additional overheads and results in a very smooth communication. To determine the size of the packets, Wireshark [19] is used. In the topology depicted in Fig. 2, a controller shares less than 1 KB data with other controllers, decreasing the overheads to a great extent and helping the topology to scale and evolve without worrying about the footprint of control information.

Considering the example in Fig. 2, every domain utilizes 3 switches and each switch represents a network. As the user configures the DMC domain, the Control Channel passes the network address information (i.e., internal address) and also shares the edge address connecting to the neighbor of the respective neighboring domains. In the topology depicted in Fig. 2, three network addresses (there are 3 networks in a domain) and one edge address is passed to every connected domain.

Fig. 5 represents the results of the Packet Exchange from one domain to another. Also, it displays the amount of information in bytes to another domain. DMC reduces the overhead to a great extent and doesn't cause flow overhead in the controller to controller communication. DMC promises to dedicate controller for its domain while offering multi-

controller communication. Each controller in a domain shares their network addresses and connecting edge address to the neighboring domain, and this information regarding message size is depicted in Fig. 5. For further explanation, since each domain has three different networks in the scenario presented in Fig. 2, the first three bars in Fig. 5 denotes the network addresses of each network listed under the domain, and final bar represents the edge network address connected to the adjacent domain. Total overhead of each message is 220 bytes, consisting 88 bytes of header and 122 bytes of the content body.

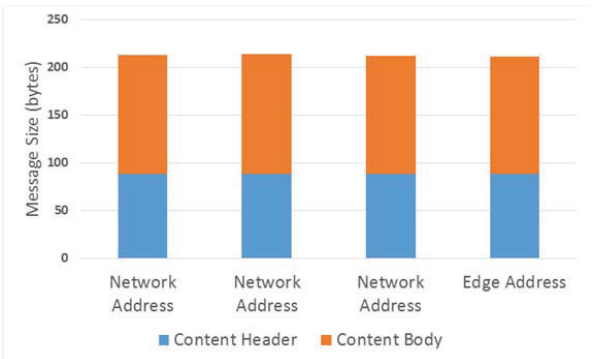

Fig. 5. Message size of control packets from one domain to another

3) Link Recovery:

This experiment test how the monitoring module reacts to the link failure and computes an alternative route in minimal time. The TCP traffic is generated using iPerf [18] between host A of Domain A and Host B of Domain B, as shown in Fig. 2. The link was disconnected at time t = 2.2 s by the using "link down" command. To determine the link recovery time, packets are inspected using Wireshark [19] on both hosts. As the link goes down, the monitoring module senses that, immediately computes an alternative route and pushes it to the switch. As shown in Fig. 6, the link was up again by t = 3.7 s, hence the link recovery time was 1.5 s.

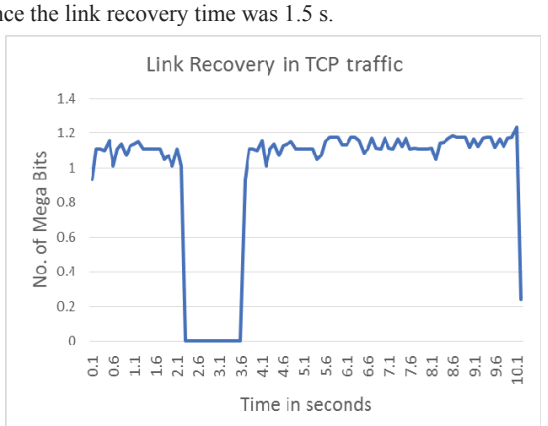

Fig. 6. Time taken for link recovery

V. Conclusion

This paper proposes a Distributed approach in Multi-domain Controllers. The proposed model interconnects multiple domains in a resilient, scalable and secure manner. The DMC relies on the centralized controller in each domain that exchanges minimal information across other controllers to provide end-to-end network services. It implements a light-weight control channel that reduces the additional overheads by sharing only the host information to other domains. Unlike other approaches, it secures privacy as the controller is solely confined to the view of its own domain. The approach is implemented on top of the RYU control platform [11] and has been evaluated for inter-domain link failure.

Future work would include the addition of a master/slave architecture within a domain, as that would be beneficial in the case of a controller failure. If a controller fails within a domain, the slave controller takes over and continues exchanging information with neighboring controllers, and this would result in a smooth communication throughout the entire network.

Acknowledgement

This research was supported in part by the Ministry of Science and Technology of Taiwan and National Chiao Tung University under Grants: MOST 104-3115-E-009-007, MOST 104-2221-E-009-023, MOST 104-2622-8-009-001, and ICTL-105-Q707.

References

[1] M.-K. Shin, K.-H. Nam and H.-J. Kim "Software-defined networking (SDN): A reference architecture and open apis," in *Proc. International Conference on ICT Convergence* (ICTC), 2012, Korea, pp. 360-361.

[2] "Software-Defined Networking: The New Norm for Networks." [Online]. Available: http://www.opennetworking.org/

[3] "Open Networking Foundation (ONF)." [Online]. Available: http://www.opennetworking.org/

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

[5] A. Tootoonchian and Y. Ganjali, "Hyperflow: a distributed control plane for openflow," in *Proc. INM/WREN*, pp. 3-3,2010.

[6] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, S. Shankar "Onix: a distributed control platform for large-scaleproduction networks," in *Proc. OSDI*, pp. 351-361, 2010.

[7] A.-W. Tam, K. Xi, and H. Chao, "Use of devolved controllers in datacenter networks," in *Proc. IEEE INFOCOM workshops* pp. 596-601, 2011.

[8] S. H. Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proc. HotSDN*, pp. 19-24, 2012.

[9] "RabbitMQ." [Online]. Available: http://www.rabbitmq.com

[10] "AMQP." [Online]. Available: http://www.amqp.org

[11] "Ryu OpenFlow Controller." [Online]. Available: http://osrg.github.io/ryu/

[12] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, "Feature-based comparison and selection of Software Defined Networking (SDN) controllers," in *Proc. 2014 IEEE World Congress on Computer Applications and Information Systems* (WCCAIS), pp. 1-7, 2014.

[13] A. Dixit, F. Hao, S. Mukherjee, T.V. Lakshman and R. Kompella "Towards an elastic distributed sdn controller" in *Proc. 2nd ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 7-12, 2013.

[14] P. Lin, J. Bi, H. Hu "ASIC: An Architecture for Scalable Intra-domain Control in OpenFlow" in *Proc. 7th international conference on Future Internet Technologies*, pp. 21-26, 2012.

[15] S. Zerrik, M. Bakhouya, J. Gaber "Towards a Decentralized and Adaptive Software Defined Networking Architecture," in *Proc. Fifth International Conference of Next Generation Networks and Services* (NGNS), pp. 326-329, 2014.

[16] K. Phemius, M. Bouet and J. Leguay, "Disco: Distributed multi-domain SDN controllers" in *Proc. 20th IEEE netw. Oper. Manag. Symp.*, pp. 1-4, 2014

[17] "Mininet." [Online]. Available: http://mininet.org

[18] "IPerf." [Online]. Available: https://iperf.fr/

[19] "Wireshark." [Online]. Available: https://www.wireshark.org/

[20] B.Astuto, M.Mendonca, X.Nguyen, K.Obraczka and T.Turletti, "A Survey of Software-Defined Networking:Past, Present and Future of Programmable Networks," *IEEE Communication Survey and Tutorials*, vol. 16, no. 3, pp. 1617-2014, 2014.