Orion: A Hybrid Hierarchical Control Plane of Software-Defined Networking for Large-Scale Networks

Yonghong Fu[1,2,3], Jun Bi[1,2,3], Kai Gao[1,2,3], Ze Chen[1,2,3], Jianping Wu[1,2,3], Bin Hao[4]
1. Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China
2. Department of Computer Science, Tsinghua University, Beijing, China
3. Tsinghua National Laboratory for Information Science and Technology, Beijing, China
4. H3C Telecommunication Technology Company, Beijing, China
Email: {fuyh11@mails.tsinghua.edu.cn, junbi@tsinghua.edu.cn}

***Abstract*—The decoupled architecture and the fine-grained flow control feature of SDN limit the scalability of SDN network. In order to address this problem, some studies construct the flat control plane architecture; other studies build the hierarchical control plane architecture to improve the scalability of SDN. However, the two kinds of structure still have unresolved issues: the flat control plane structure can not solve the super-linear computational complexity growth of the control plane when SDN network scales to large size; the centralized abstracted hierarchical control plane structure brings path stretch problem. To address the two issues, we propose Orion, a hybrid hierarchical control plane for large-scale networks. Orion can effectively reduce the computational complexity growth of SDN control plane from super-linear to linear. Meanwhile, we design an abstracted hierarchical routing method to solve the path stretch problem. Further, Orion is implemented to verify the feasibility of the hybrid hierarchical approach. Finally, we verify the effectiveness of Orion both from the theoretical and experimental aspects.**

I. Introduction

In Software Defined Networking (SDN), the control and data planes are decoupled, and the complex control and management functions are stripped out of the network device[1]. Meanwhile, SDN supports a flow-based management to enable highly programmable and flexible of Networks. OpenFlow switches are mainly used to achieve the fine-grained flow control in SDN[2]. However, such decoupled architecture and fine-grained flow control feature bring a large amount of communication messages between the data plane and the control plane which limit the scalability of the SDN network [3, 4, 5].

The scalability of Software-Defined Networks is so important that many researchers have been trying to solve this problem. Maestro exploits parallelism in single-threaded controller[6]. Beacon employs multi-threaded technics to improve the scalability of a single controller[7]. But the two studies [6] and [7] do not yet provide communication between multiple controllers. DevoFlow considers the SDN controller handling too many micro-flows, which creates excessive load on the controller and switches [4]. Then it proposes a way that the control plane maintains a useful amount of visibility without imposing unnecessary costs. DIFANE employs authority switches to store necessary rules to share the work load of the control plane[8]. However, both studies [4] and [8] require to modify the OpenFlow switch. Then some researchers design different control plane structures to extend the control planes processing ability. On one hand, some studies construct the flat control plane architecture, such as HyperFlow, Onix and ONOS. HyperFlow presents a distributed event-based control plane for OpenFlow[3]. Onix develops a distributed system, which runs on a cluster of one or more physical servers[9]. ONOS is an experimental open source distributed SDN OS provides scale-out SDN control plane and achieves a global network view in a scaled-out fault tolerant network OS [10]. On the other hand, some studies build the centralized hierarchical control plane of SDN, such as Kandoo and Logical xBar. Kandoo has a two layer hierarchical architecture. The bottom layer controllers run local control applications based on local network view, and the top layer controllers run global applications based on global network-wide view [11]. Further, Logical xBar introduces a recursive building block to construct abstracted hierarchical SDN networks [12].

Though the above two kinds of control plane architecture can improve the scalability of SDN networks, they still have unresolved issues. **1) The flat control plane architecture can not solve the super-linear computational complexity growth of the control plane when SDN network scales to large size**. We argue that the fine-grained flow control feature of SDN will lead to the super-linear computational complexity growth of the control plane and that limits the scalability of SDN networks. We take an example to illustrate the problem. Assuming that a SDN controller manages M network devices; it uses the <SrcIP, DstIP> to identify the data flow; the SDN controller adopts Dijkstra algorithm to compute routing paths. In the beginning, the computational complexity of the Dijkstra algorithm is $O(M^2)$. When the network size increases N times, there will be $N * M$ nodes. Then the computational complexity of the routing algorithm increases to $O(N^2M^2)$. If we use N SDN controllers to share the work load of the

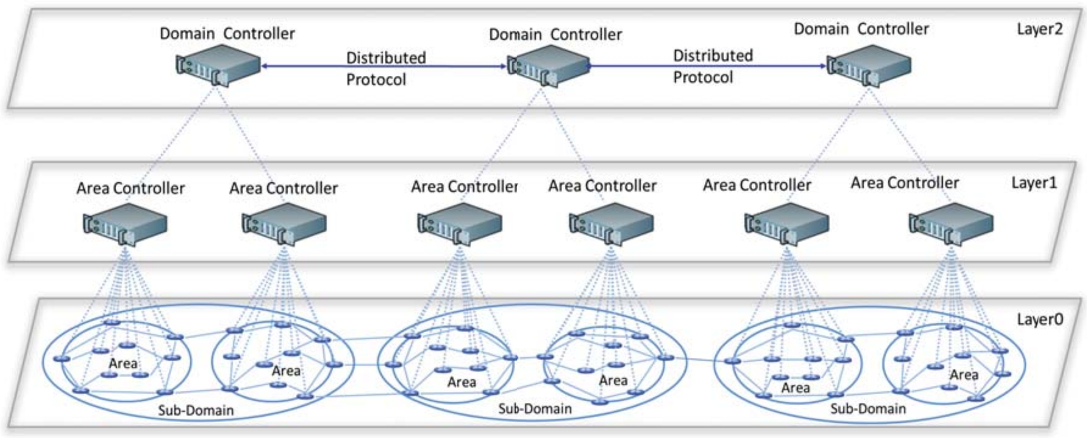


Fig. 1. Orion's Hybrid Hierarchical Architecture

control plane, the control plane processing capacity increases N times, but the computational complexity of the control plane increases N^2 times. **2) The centralized abstracted hierarchical control plane architecture brings path stretch problem**. In a network graph, a Stretch(u,v) is the stretch of path from node u to node v. It is defined as $Stretch(u,v) = \frac{Path(u,v)}{ShortestPath(u,v)}$, where Path(u,v) is the length of the path from node u to node v and the ShortestPath(u,v) is the shortest length of the path from node u to node v. The centralized abstracted hierarchical control plane, such as Logical xbar, can solve the super-linear computational complexity growth of the control plane by constructing abstracted hierarchical network views. However, the more layers it abstracts, the bigger the path stretch is.

In order to address the above two problems, we propose Orion, a hybrid hierarchical control plane for large-scale networks. The proposed hybrid hierarchical control plane architecture is relative to the flat control plane architecture (such as HyperFlow and Onix) and the centralized hierarchical control plane architecture (such as Kandoo and Logical xBar). It makes the following contributions: We design Orion, a hybrid hierarchical control plane. Orion can reduce the computational complexity growth of SDN control plane from super-linear to linear by constructing abstracted hierarchical network views (introduced in the section II). Thus, the computational complexity of the control plane is reduced. Meanwhile, we design an abstracted hierarchical routing method to reduce the path stretch problem based on the architecture of Orion (introduced in the sub-section C of section II). Further, we implement Orion to verify the feasibility of the hybrid hierarchical approach, and we verify its effectiveness both from the theoretical and experimental aspects.

The rest of this paper is organized as follows. We introduce the design of Orion in section II. Next, the implementation and evaluation of Orion are presented in Section III. Finally, conclusions are given in Section IV.

II. Design

In this section, we present the design of Orion, a hybrid hierarchical control plane architecture of SDN. Orion focuses on the intra-domain control and management of large-scale networks. In this paper, domain is a whole network which can be controlled and managed by one administrator. It divides into several sub-domains. A sub-domain consists multiple areas that located relatively close to each other. Area refers a region that can be controlled by a single SDN controller.

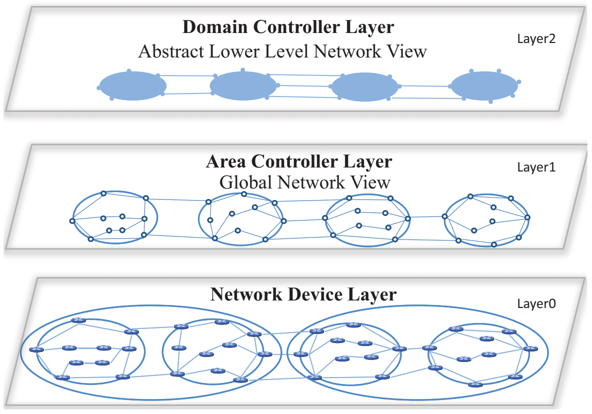


Fig. 2. Network View of the Hybrid Hierarchical Architecture

A. Architecture

The hybrid hierarchical architecture of Orion is shown in Fig. 1. Orion has three layers. 1) The bottom layer of Orion is the physical layer. The physical layer is composed of large amounts of connected OpenFlow switches. 2) The middle layer of Orion is the area controller layer. The area controller is responsible for collecting physical device information and link information, managing the intra-area topology and processing intra-area routing requests and updates. Meanwhile, it abstracts its area network view and sends it to the top layer. 3) The top layer of Orion is the domain controller layer. In this layer, the domain controller treats area controllers as devices, and it synchronizes the global abstracted network view through a distributed protocol.

Through dividing areas and constructing abstracted hierarchical views, Orion can reduce the computational complexity of the control plane from super-linear growth to linear (illustrated in the sub-section D of section II). We show the hierarchical network views of Orion in Fig. 2. The bottom layer's view is the view of physical device Layer. The middle layer's view is the view of area controller Layer. The top layer's view is the view of lower abstracted area view. In this layer, every area is abstracted as a node, and the exports of the area are regarded as the ports of the node.

B. Components

There are eight major components in Orion, shown in Figure 3. Among these components, the Host Management, Topology Management, Routing, Storage and Vertical Communication Module have two sub-modules. The two sub-modules are responsible for intra-area information processing and inter-area information processing.

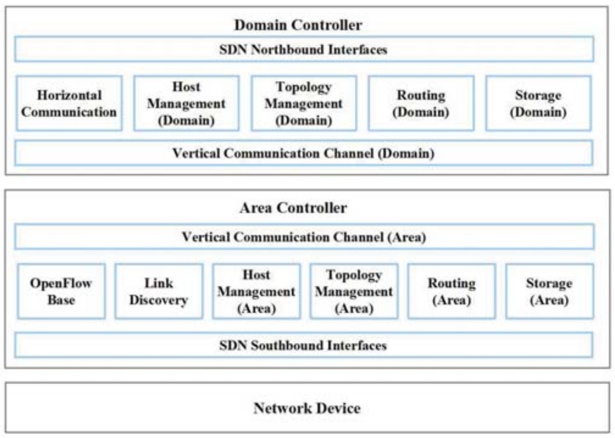


Fig. 3. The components of Orion

OpenFlow Base Module. The OpenFlow Base Module is responsible for collecting OpenFlow switch information and receiving messages through the SDN southbound Interfaces. Meanwhile, it provides an interface for the area controller to install rules on OpenFlow switches, such as Flow-Mod or Packet-Out.

Host Management Module. The Host Management Module has two parts to deal with the area host information and domain host information. 1) The Area Host Management Sub-Module obtains the host information in its area through the ARP packet sent by the host. 2) The inter-area host information is managed by the Domain Host Management Sub-Module. It works as an ARP Proxy in Orion. When there are multiple areas and multiple controllers, two hosts in different areas do not know each other's specific information, such as MAC address. So the source host sends an ARP packet to ask the MAC address of the destination host. When the ARP packet reaches the switch which connect to the source host, the switch sends a Packet-In message to the area controller. Then the area controller checks whether it has the destination host information or not. If the area controller does not have the information, it sends the ARP packet to the domain controller. The domain controller checks the information of the destination host in the database. If the domain controller gets result from the database, it sends the result to the area controller. Then the area controller sends the result back to the source host. If the domain controller gets nothing from the database, it broadcasts the ARP packet to all the ports of every switch in its sub-domain. In the process of the broadcast, in order to avoid broadcast storm, we use an algorithm similar to Spanning Tree to avoid broadcast loop.

Link Discovery Module. The link discovery module obtains the intra-area link information through LLDP protocol. In order to obtain inter-area link information, an area controller sends LLDP packets to all ports of its edge switches with its ControllerID. When the LLDP packet reaches the edge switch in another area, the switch encapsulates the LLDP packet into a Packet-In message and sends the message to its area controller. Then the area controller decapsulates the Packet-In message and gets the TLV message out of the LLDP Packet. If the LocalControllerID in the LLDP message is different from its own ControllerID, then it knows that the link is an external link and the switch that sends the Packet-In message is an edge switch in its area.

Topology Management Module. The Topology Management module has two sub-modules. 1) The Area Topology Management sub-module manages the physical topology information received from the Link Discovery Module. Meanwhile, it computes the shortest path from every edge switch to other edge switches. Then it sends the switch information and the hops between any two edge switches to the domain controller. When the domain controller receives the above information, it treat an area as a node, and treat the edge switches of the area as a port, the hop between any two edge switches is like the weight of an abstract link between two port. 2) The Domain Topology Management sub-module is used to manage the abstract topology of the top layer.

Storage Module. The Storage Module includes three kinds of information: host information, switch information and link information. Host information includes <MAC address, SwitchID, SwitchPort, LocalControllerID>. Switch information includes <SwitchID, SwitchPort, PortStatus, LocalControllerID,IsEdgeSwitch>. The link information includes abstract link information and real physical link information. There are two kinds of abstract link. The first kind of abstract link is the link from every inner switch to all edge switches in each area. The abstract link here means the shortest path hops from the inner switch to the edge switch. The second abstract link is the link from every edge switch to other edge switches in each area, and the abstract link here means the shortest path hops from one edge switch to another edge switch. The physical link indicates the real physical inter-area links between different areas. In the Area Storage Sub-Module, it stores the host information, switch information and real physical link information of the area. In the Domain Storage Sub-Module, it stores the host information, switch information and the abstract link information of all areas.

Horizontal Communication Module. The Horizontal Communication Module is responsible for the synchronize global abstract Network Information among the domain controller cluster. In this part, we use a scalable NoSQL database which supports dynamic clustering to store global host information, global switch information and global abstract topology information.

The distributed policy. The distributed policy of the Horizontal Communication Module mainly includes three aspects: Unique ControllerID, Publish/Scribe and Local Cache. 1) Unique ControllerID. The domain controller use distributed lock method to guarantee the unique controller ID for itself. Meanwhile, the domain controller is responsible for generating unique ID for all the area controllers it connects. 2) Publish/Scribe. The domain controllers use distribution mechanism to publish and scribe messages (such as distributing routing rules). With this mechanism, domain controller can create topic, and other domain controllers who are interest in the topic, they can subscribe the topic. 3) Local Cache. The global domain information (including all hosts, switches and topology information) are partitioned across the domain controller cluster. If we want to read a key from the NoSQL database multiple times, and the key is owned by another domain controller. Then we will cost a lot for every time we read a remote fetch. So we use cache strategy to store the keys with high visit frequency to improve reading speed and cost less network bandwidth.

Distribution of Rules. The distribution of routing rules is realized through the Publish/Scribe mechanism. In the top layer of Orion, every domain controller creates a topic with its ControllerID. When two hosts communicates between two different sub-domains, a flow routing computation request is sent to the domain controller which is responsible for the source sub-domain. For the domain controller has global abstract topology, so the source domain controller calculates the overall inter-area routing path. Then it gets the ControllerID of all area controllers which own the edge switches on the routing path, and gets the ControllerID of the corresponding domain controllers. Then the source domain controller publishes the routing rules to the topic of all domain controllers on the routing path according to their ControllerID obtained above. Then the domain controllers sends the routing rules to the area controllers according to the received area ControllerID through the Vertical Communication Module introduced below.

Vertical Communication Module. The vertical communication channel between the area controllers and the domain controllers is established via the TCP connection. It is used to send requests and distribute rules. 1) Send Requests. The vertical communication channel between area controllers (client) and the domain controllers(server) is established via the TCP connection. In this part, the domain controller is built with asynchronous socket. The asynchronous socket creates a thread pool during initialization, and using a thread polling to see if there is a new message coming. When an area controller sends a message to the domain controller it connect with, the socket in the server gives the message to the thread pool. Then the thread pool calls the relevant modules to handle the incoming message. 2) Distribute Rules. The domain controller distributes rules and ARP replies through the established TCP connection with the area controller. When the domain controller distributes routing rules to relative switches, it sends rules in a reverse sequence manner. That means the first distribute area is the area connects with the destination host, and the last distribute area is the area connects with the source host. The reverse distributes method is used to prevent one same data flow generates several Packet-In messages in different areas.

Failover. The fault tolerance and failover between switches and area controllers is guaranteed by OpenFlow protocol. From OpenFlow v1.2, controllers have their roles, such as Master, Slave or Equal. When a Master area controller fails, the Slave area controller can quickly take all the switches that connect to the failure Master controller. Meanwhile, the fault tolerance and failover between area controllers and domain controllers reference the role mechanism of OpenFlow v1.2.

Routing Management Module. This module includes t-wo sub-modules: Area Routing Management sub-module and Domain Routing Management sub-module. In this part, we design a routing algorithm which can effectively reduce the path stretch problem of the hierarchical structure. The detailed design of it is introduced in the following sub-section C of Section II.

C. Abstracted Hierarchical Routing Method

As we know the abstracted hierarchical control plane architecture brings the path stretch problem. An abstracted hierarchical routing method is designed to address the problem. The abstracted hierarchical routing method is based on the Dijkstra algorithm[16]. Dijkstra algorithm is a graph search algorithm widely used in network routing protocols, such as IS-IS[17] and OSPF[18]. The core idea of the abstracted hierarchical routing method is similar to IS-IS and OSPF. Through dividing the whole network into multiple areas, the administration complexity and routing computational complexity is reduced. But IS-IS and OSPF protocols are running on traditional distributed routers, the abstracted hierarchical routing method runs on the centralized area controllers and distributed domain controllers.

The abstracted hierarchical routing method is divided into two parts. 1) As the area controller has the detailed inner information of its area, the Area Routing Management Sub-Module of the area controller pre-computes the inner hops from every inner switch to all edge switches by Dijkstra algorithm and send the result to the domain controller. 2) The Domain Routing Management Sub-Module of the domain controller computes the global shortest path. Though the domain controller level only has the abstracted lower level network view, it can compute the shortest path for the flow based on the sum of the inner path result sending by the area controller and the inter-area path length.

Area Routing Management Sub-Module. When a Packet-In message reaches the area controller, the area routing management sub-module checks the source address and destination address of the message. 1) If the destination address is in the area, the area controller employs Dijkstra algorithm to compute intra-area path. 2) If the destination address is out of the area, the area controller sends the source address and the destination address of the message to the domain controller, and store the message to a waiting buffer with index. The domain controller computes the routing path for the flow and sends the routing result (<IngressSwitchID, IngressPort, EgressSwitchID, EgressPort>) to the area controller. After the area controller receives the result, it sends the buffered message to the exports of its original switch. Meanwhile, it calculates the intra-area path for the data flow and installs flow entries in all switches on the routing path. Meanwhile, the area controller needs to calculate the routing path for data flows passes its area. When there is a flow go through the area of an area controller, the top layer domain controller sends the routing message <IngressSwitchID, IngressPort, EgressSwitchID, EgressPort> to the area controller. Then the area controller calculates the intra-area routing path for the data flow. Further, in order to address the path stretch problem, the area controllers pre-computes the inner hops from every inner switch to all edge switches and sends the result (<InnerSwitchID, EgressSwitchID, hops> of all inter-area inner switches) to the domain controller it connects.

Domain Routing Management Sub-Module. At first, the domain routing management sub-module uses Dijkstra algorithm to calculate the inter-area routing path. Next, it collects the intra-area hops from the inner switch to all edge switches which is sending by area controllers, and adds the inter-area hops and the intra-area hops together. The shortest length path determining the final forwarding path. At last, the Domain Routing Management sub-module sends the routing message <IngressSwitchID, IngressPort, EgressSwitchID, EgressPort> to all area controllers on the routing path. Then every area controller calculates the intra-area routing path in its area and installs forwarding rules on all switches on the routing path. Thus the domain controller level can compute global shortest path, the value of Path(u,v) is equals to the value of Shortest-Path(u,v), then $Stretch(u, v) = \frac{Path(u,v)}{ShortestPath(u,v)} = 1$. Thus Orion reduces the path stretch value to 1.

Routing Example. We give two examples to illustrate how Orion carries out the intra-area routing and inter-area routing. The two routing examples are based on the topology shown in Fig. 4. In the topology, a domain has two sub-domains and each sub-domain has two areas. At the same time there are four host (host A, B, C and D) located in the topology.

Intra-area Routing Example. The first example is an example of the intra-area routing. This example shows how to communicate between two hosts (such as host A and host B) in an area. When the host A sends a data flow to the host B, the switch which connect to the host A generates a Packet-In message and sends the message to area controller1. When area controller1 receives the message, it checks whether the destination address of the data flow is in its area. As host B is located in area1, so area controller1 can find the information of host B. Then it calculates the intra-area routing path from host A to host B based on the intra-area topology. Next, area controller1 sends the routing rules to the switches in the path list, so that the switches can install the rules for the data flow. Finally, when all the switches in the path list are installed routing rules, the data flow sent by host A is forwarded to host B. The intra-area routing sequence of Orion is shown in Fig. 5.

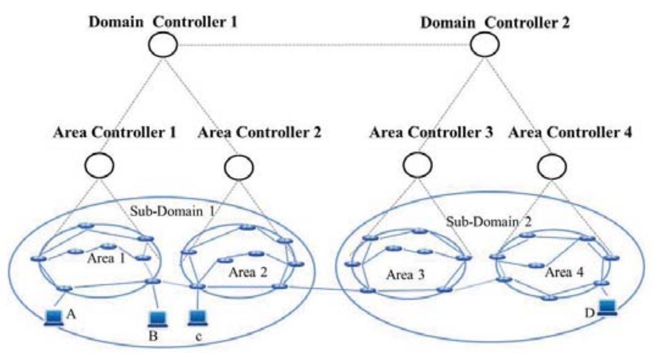


Fig. 4. Example topology

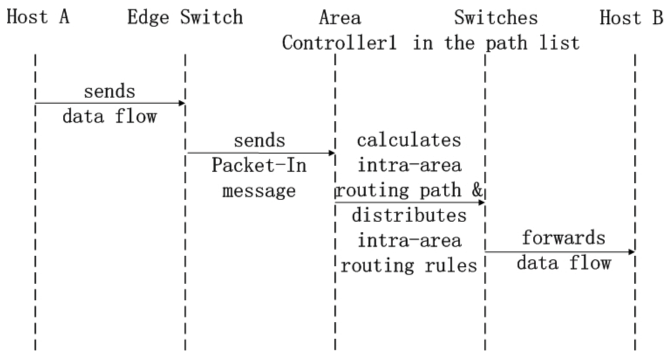
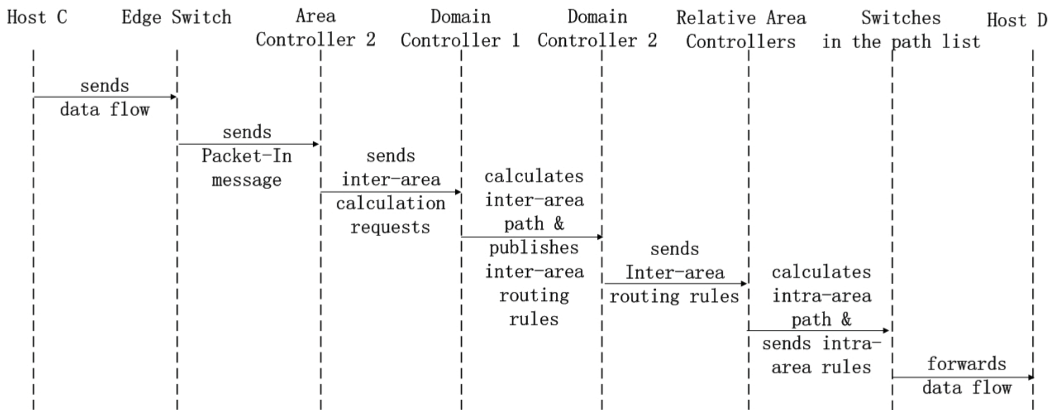

Host A
Edge Switch
Area Controller1
Switches in the path list
Host B
sends data flow
sends Packet-In message
calculates intra-area routing path & distributes intra-area routing rules
forwards data flow

Fig. 5. Intra-area routing sequence of Orion

Inter-area Routing Example. The second example is an example of the inter-area routing. The example illustrates how host C sends data flow to host D with Orion. When host C sends a data flow to host D, the data flow reaches the switch which host C connects to. Then the switch generates a Packet-In message and sends the message to area controller2. As host D is not in area2, when area controller2 receives the message, it extracts the <source address, destination address> from the Packet-In message and encapsulate it to a simple request, sends the request to domain controller1, and buffers the Packet-In message with an index. When domain controller1 receives the request, it calculates the inter-area path according to



Host C
Edge Switch
Area Controller 2
Domain Controller 1
Domain Controller 2
Relative Area Controllers
Switches in the path list
Host D
sends data flow
sends Packet-In message
sends inter-area calculation requests
calculates inter-area path & publishes inter-area routing rules
sends Inter-area routing rules
calculates intra-area path & sends intra-area rules
forwards data flow

Fig. 6. Inter-area routing sequence of Orion

the global abstracted network view. For the destination host D is in the sub-domain of domain controller2, so domain controller1 publishes the inter-area path routing rules to the domain controller2 through the NoSQL distributed database. When domain controller2 receives the inter-area routing rule messages, it sends the messages to the area controllers on the routing path. When area controller3 and area controller4 receives the messages, they analyze the messages and know the <IngressSwitch, IngressPort, EgressSwith, EgressPort> of the data flow. Then they calculate the intra-area routing path for the flow and sends the routing rules to the switches on the routing path. Finally, when all switches on the routing path installed routing rules, the data flow is forwarded from host C to host D. The inter-area routing sequence of Orion is displayed in Fig. 6.

D. Module Relationships

In the eight major components of Orion, the OpenFlow Base Module is the base module responsible for collecting OpenFlow switch information and receiving messages through the SDN southbound Interfaces. It provides interfaces for the Link Discovery Module and the Area Host Management Sub-Module of the Host Management Module to let them obtain physical link information and host information. The Area Host Management Sub-Module sends the area host information to the Domain Host Management Sub-Module. The Area Topology Management Sub-Module of the Topology Management Module manages the physical topology information received from the Link Discovery Module. Meanwhile, it sends the intra-area abstracted topology to the Domain Topology Management Sub-Module through the Vertical Communication Channel Module. The Area/Domain Storage Sub-Module s-tores the host information and topology information received from the Area/Domain Host Management Sub-Module and the Area/Domain Topology Management Sub-Module, and provides local/globle host, switch and link information for the Area/Domain Routing Sub-Module. The Horizontal Communication Module calls the Domain Storage Sub-Module to synchronize the global abstracted network view.

E. Benefits and Tradeoffs

Benefits. The hybrid hierarchical architecture can efficiently reduce the control plane's computational complexity growth from the super-linear to linear when network scales to large size. Assuming there are N areas, and each area has M_i nodes. To facilitate the discussion, we assume that $M_i = M$. Then the computational complexity of every independent area controller is $O(M^2)$. For there are N area controllers in the area controller layer, then the total computational complexity of the area controller layer is $N * O(M^2)$. As the domain controller calculates paths based on the abstracted network view, when there are N area controllers in the lower layer, the computational complexity of every domain controller is $O(N^2)$. Assuming there are c domain controllers in the top layer. Then, the computational complexity of the domain controller layer is $c * O(N^2)$. So the total computational complexity of the control plane is $N * O(M^2) + c * O(N^2)$. Thus, we can see that the computational complexity of the control plane is reduced from super-linear growth to linear growth.

Tradeoffs. In order to address the path stretch problem, the abstracted hierarchical routing method needs the area controller pre-computes the inner hops from every inner switch to all edge switches and sends the result to the domain controller. The computation complexity of the pre-compute process is $O(M_i^3)$, where M_i is the number of nodes in the area. Then the computation complexity of this method in all areas is $Max(O(M_i^3)), i = 1, 2, ..., N$, where N is the number of areas. Meanwhile, the computational complexity of the abstracted routing approach is related to the number of edge switches. If there are K edge switches in the domain, the computation complexity from an edge switch to another edge switch is $O(K^2)$. The computation complexity from one inner switch in an area to another edge switch is $O((K+1)^2)$, and the computation complexity from one inner switch in an area to another inner switch in another area is $O((K + 2)^2)$.

III. Implementation and Evaluation

In this section, we present the implementation and evaluation of Orion. In this part, we evaluate Orion through both theoretical and experiment ways.

A. Evaluation

1) Theoretical Evaluation: We write a simple single-threaded Dijkstra algorithm to calculate the path from the source address to the destination address. The algorithm is running under random topology. In the random topology, if there are N nodes in the topology, then the topology has N^x edges, where x is a variable. We run the algorithm on a server with Intel E5645 processors (6cores in total, 2.40GHz) and 64GB memory. We compare the proposed abstracted hierarchical routing method with the traditional Dijkstra algorithm.

Under the best conditions, the computational complexity of the abstract hierarchical routing method is $O(K^2)$, where K is the number of edge switches. The best conditions refer that the network does not change, the area controllers do not need to re-compute the inner hops from every inner switch to edge switches. Meanwhile, the source host and destination host of the flow are both connected to edge switches. In this experiment, we assume that the number of edge switches accounts for 10%, 20% and 50% of all switches in the domain. In this experiment, we choose $x = 1.25$ and $x = 1.15$. The total number of switches N ranges from 0 to 3000, and the number of switches in an area M is 100. From Fig.7, we can observe that with the increasing number of areas, the computing time of Orion is increased as linear growth, much lower than the traditional Dijkstra routing algorithm.

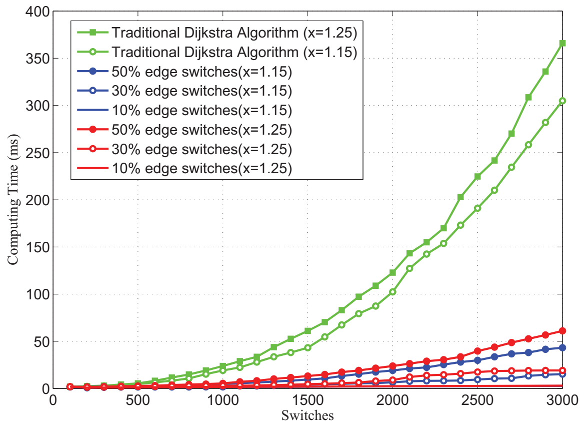


Fig. 7. Theoretical evaluation under the best condition

Under the worst circumstances, the computational complexity of this method is $Max(O(M_i^3)) + O((K + 2)^2)$. The worst conditions refer that the network changes dramatically; area controllers need to re-compute the inner hops from every inner switch to all edge switches; the source host and the destination host of a flow are both connect to inner switches in two different areas. The parameters of this test are same with the above experiment. From Fig.8, we can note that when the network nodes exceeds 2900, the abstracted hierarchical routing method proposed by Orion is better than the traditional Dijkstra algorithm.

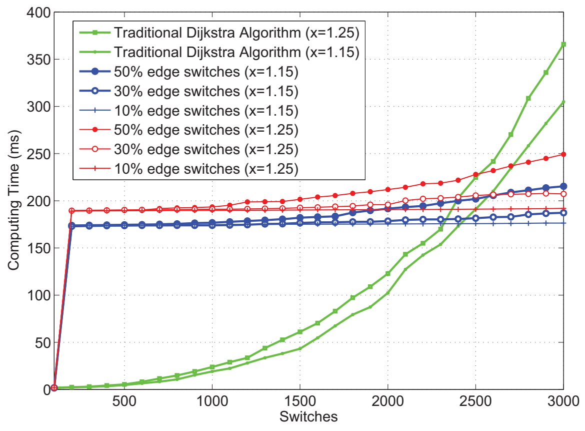


Fig. 8. Theoretical evaluation under the worst condition

2) Experiments Evaluation: In this part, we build a prototype system to verify the feasibility and effectiveness of Orion. The implementation of Orion is in Java. The area controller of Orion is build based on the Floodlight controller [19]. The intra-area OpenFlow Base module, Link Discovery module and Storage module are based on Floodlight. We extend the Floodlight controller to construct the other modules. The domain controller of Orion is not an SDN controller with OpenFlow protocol. The communication between the domain controller and the area controller is through the Vertical Communication Module of Orion, and it is not based on OpenFlow protocol. This communication method improves the domain Controller's performance and reduces the communication messages. We run Orion on a server with two Quad-Core Intel $E5-2650$ processors (16 cores in total, 2.00GHz) and 128GB memory. Through virtualization, the whole server is divided into multiple virtual controllers. Every controller in Orion has 2 Core, 8GB memory. Meanwhile, we use the Mininet [20] to simulate the data plane.

In the fist experiment, each area has 120 switches. Meanwhile, the network topology is using the random network topology ($x = 1.25$). Due to the hardware and bandwidth limitation, we only start two domain controllers and six area controllers. In this test, we increases the number of areas to test the flow set-up rate per second of Orion. From Fig. 9, we can see that when there are only one area, the area controller can handle 8126 new flows per second. With the increasing number of areas, the average throughput of the control plane is stable, and the overall throughput of Orion increases steadily.

In the second experiment, we test the delay between areas. The number of switches in each area M ranges from 20 to 120. The number of domain controllers increases from 1 to 2. The number of areas is varying from 2 to 6. From Fig. 10,

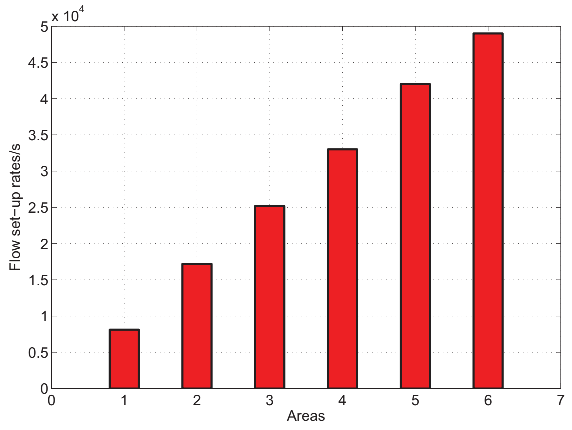


Fig. 9. Flow set-up rate of Orion

we can see that with the increasing number of areas, the delay time gradually increased.

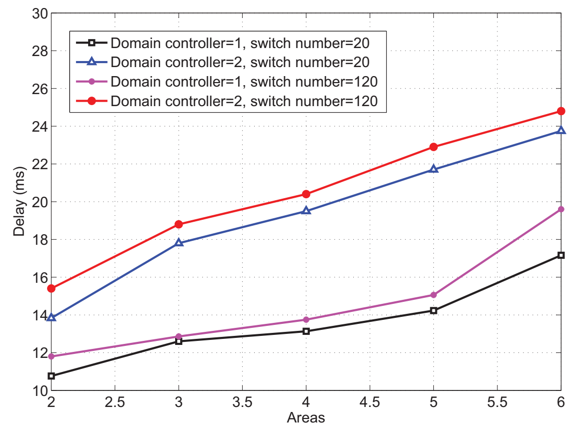


Fig. 10. The average delay time of Orion

Overheads. Orion does not introduce much overhead. We monitor the overhead of Orion when we test its throughput. In the throughput test, the inter-area flows are more than inter-area flows. The CPU utilization of the domain controller is 40%. For the domain controller of Orion is just a simple server, it cost much lower than an OpenFlow based SDN controller. The load of the area controller of Orion is between the load of the controller in the flat architecture and the lowest area controller in the abstracted hierarchical SDN architecture.

IV. Conclusion

In this paper, we design and implement Orion, a hybrid hierarchical control plane for large-scale networks. Orion addresses the super-linear computational complexity growth of the control plane when SDN network scales to large size, and solves the path stretch problem brought by the abstracted hierarchical control plane architecture. Further, we evaluate the effectiveness of Orion through theoretical and experiment aspects. Our evaluation results show the efficiency and feasibility of Orion.

Acknowledgment

This research is supported by the National High-tech R&D Program (”863” Program) of China ($No.2013AA013505$), the National Science Foundation of China ($No.61161140454$), National Science & Technology Pillar Program of China ($No.2012BAH01B01$), and the National Science Foundation of China ($No.61303194$). In addition, we would like to thank Baobao Zhang, Yangyang Wang, Bingyang Liu and Guang Yao for some very good comments on this work. Jun Bi is the corresponding author.

References

[1] ONF White Paper, Software-Defined Networking: The New Norm for Networks, Open Networking Foundation, 2012.
[2] 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, Vol.38, No.2, pp.69-74, 2008.
[3] A. Tootoocian and Y. Ganjali, HyperFlow: A distributed control plane for OpenFlow, In Proc. ACM INM/WREN, 2010.
[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, DevoFlow: Scaling Flow Management for High-Performance Networks, In Proc. ACM SIGCOMM, 2011.
[5] J. McCauley, A. Panda, M. Casado, T. Koponen, S. Shenker. Extending SDN to Large-Scale Networks, In Proc. ONS, 2013.
[6] Z. Cai, A. L. Cox, and T. S. E. Ng, Maestro: A System for Scalable OpenFlow Control, Technical Report, Rice University, 2010.
[7] D. Erickson, The Beacon OpenFlow Controller, In Proc. ACM SIGCOM-M HotSDN, 2013.
[8] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, Scalable Flow-Based Networking with DIFANE, In Proc. ACM SIGCOMM, 2010.
[9] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, Onix: a distributed control platform for large-scale production networks, In OSDI, 2010.
[10] B.Lantz, B. Connor, J. Hart, P. Berde, P. Radoslavov, M. Kobayashi, T. Koide, Y. Higuchi, M. Gerola, W. Snow, G. Parulkar, ONOS: Towards an Open, Distributed SDN OS, In Proc. ACM SIGCOMM HotSDN, 2014.
[11] S. H. Yeganeh, Y. Ganjali, Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications, In Proc. ACM SIGCOMM HotSDN, 2012.
[12] J. McCauley, A. Panda, M. Casado, T. Koponen, S. Shenker, Extending SDN to Large-Scale Networks, In ONS, 2013.
[13] J. Sherry, S. Hasan, C. Scott, Making Middleboxes Someone Elses Problem: Network Processing as a Cloud Service, In Proc. ACM SIGCOMM HotSDN, 2012.
[14] M. P. Fernandez, Evaluating OpenFlow Controller Paradigms, In Proc. ICN, 2013.
[15] ONF. OpenFlow Specification v1.2. December, 2011.
[16] E.W. Dijkstra, Dijkstra: A note on two problems in connexion with graphs, In Numerische Mathematik, vol. 1, no. 1, pp. 269C271, 1959.
[17] IS-IS protocol specification (IETF), RFC 1142, [Online]. Available: http://tools.ietf.org/html/rfc1142.
[18] OSPF Version2, RFC 2328, [Online]. Available: https://tools.ietf.org/html/rfc2328.
[19] Floodlight, http://www.projectfloodlight.org/floodlight/
[20] B. Lantz, B. Heller, and N. McKeown, A network in a laptop: rapid prototyping for software-defined networks, In Proceedings of the ACM SIGCOMM HotNets workshop, pages 19:1-19:6, 2010.
[21] A. Dixit, F. Hao, S. Mukherjee, T.V. Lakshman, R. Kompella, Towards an elastic distributed SDN controller, In Proc. ACM SIGCOMM HotSDN, 2013.