Design and Implementation of a Scalable SDN-OF Controller Cluster

Min Luo*, Quancai Li+, Man Bo+, Ke Lin+, Xiaorong Wu+ , Chenji Li+, Sheng Lu+ , Wu Chou*
*Shannon Lab, Huawei Technologies, Inc., Santa Clara, USA
+Shannon Lab, Huawei Technologies, Inc., Hangzhou, China
Corresponding Author Email: min.ch.luo@huawei.com

***Abstract* – Software-defined networking (SDN) is a new paradigm to increase network resource utilization, minimize management complexity, and reduce operational cost. While the logically centralized control in SDN offers many unique advantages that can enable globally optimized routing and resource utilization, the controller in SDN needs to be carefully designed to avoid being a performance bottleneck or a potential single point of failure of the network. This paper presents a scalable clustering approach for the design and implementation of SDN-OF controller. In the proposed approach, controller instances (threads, and processes on a single or multiple physical servers) will mostly run in equal mode for packet-in processing, and only few such instances will serve as the "master" to process special packets that may result in updating network states. A self-learning adaptive mechanism is incorporated to further optimize the number of the controller instances under the varying operating conditions to adapt to changes in network states, controller workloads, and controller data traffic flows dynamically. As a consequence, it automatically enables load balancing and fail-over protection. The proposed approach has been successfully implemented and tested under some stringent conditions, and the performance advantages are observed.**

Keywords* – *Software defined networking; Openflow; Centralized Controller Cluster; Message Dispatching Cluster; Message Processing Cluster; Load Balancing; Fail-over.

I. INTRODUCTION

Recent advances in computing technologies have led to new business opportunities with improved business agility, high productivity, and reduced operational cost. However, while devices and applications on the network are advancing rapidly, the underlying networking infrastructure (architecture and the protocol stack specifically) has been evolving at a much slower pace. This disparity causes difficulties and ineffectiveness to support applications emerging from new business opportunities, because the traditional network control and management environment is not well-suited for new innovation, especially the complex legacy network infrastructure which cannot meet the growing demands of the application needs. There is a growing effort, spearheaded by the Open Networking Foundation (ONF) to create an open and programmable networking environment through software-defined networking (SDN) [1]. OpenFlow [2] is such an industry effort of ONF, in which the network control plane is logically centralized and decoupled from the data forwarding plane. In SDN, OpenFlow (OF) is used as the protocol between the SDN controllers and the data switches to manage and dictate the networking behavior of OF compliant switches.

In this paradigm, application developers, enterprises, and carriers can gain unprecedented programmability to control how data flows in the underlying data networks to best support the applications and readily adapt to their changing business needs. The openness and the flexibility in SDN provide new capabilities for offering better Quality of Service, and reducing the need to purchase specialized and expensive networking equipments. However, in SDN, network controller forms a logically centralized network control apparatus. If the processing power of the controller remains constant, flow setup time delay can grow in proportion to the packet-in traffics to the controller. This situation needs to be carefully addressed in SDN, especially when it scales up with an increasing number of OF switches in the network [2]. As such, there is a critical need towards a scalable centralized controller design and implementation, such that it can scale well with the size of the network and the growth of the traffics. To address this critical technical issue in SDN, new techniques and designs are needed to overcome the limitations in existing solutions. In addition, more challenging stress tests and studies on the controller scalability are needed because of the central role of the controller(s) in SDN.

SOX [9] is designed and implemented based on model driven technologies for extensibility and consitency. This allows SOX to take advantage of the recent advances in distributed computing, to enhance the controller generality, scalability, reliability, and interoperability. In addtion to be a generalized SDN controller which can support multiple evolving OF standards, e.g. OF1.0, OF1.2, OF1.3, etc., it also provides some key transformation capabilities to interworking with existing networks with BGP/PCE and exisiting transporting protocols such as MPLS.

In this paper, we present an extension to SOX, and describe a centralized SDN-OF controller cluster architecture to scale up the network management power with the size of the data network (e.g. department, campus, branch data center or even a WAN, etc.). The remaining part of this paper is organized as follows. In Section II, we briefly summarize some design and architectural principles that is applied in the proposed approach. In Section III, we describe the design and architecture of the proposed scalable controller cluster and its main functional components. In Section IV, we present scalability studies and performance results. Findings of this paper and some future R&D directions are summarized in Section V.

 ISBN: 978-1-61208-416-9

II. Related Design and Architecture

HyperFlow [3] is a distributed event-based control plane for OpenFlow, in which each controller directly manages the switches connected to it and indirectly programs or queries the other controller through proprietary communications. However, its scalability, performance, and robustness needs to be further studied and evaluated on large test-beds with more stressful conditions. ONIX [4] provides a platform in which a network control plane can be implemented as a distributed system, and it provides a general API for interaction among data, control, and management planes. It recommends that, as a good "best practice", some consistency among distributed controllers could be sacrificed for high performance requirements. However, it does not address how to leverage the advanced high performance computing techniques to improve the performance of the centralized controller in order to minimize such inconsistency. ElastiCon [5] focuses on dynamically shifting workload to allow the controllers operating at a pre-specified load window, while Kandoo [6] and DIFANE [7] try to limit the overhead of processing frequent events on the control plane. Moreover, DIFANE would keep all traffics in the data plane under control by selectively directing packets through intermediate switches with specific rules. Finally, OpenDayLight [8] currently allows a switch to be connected to one or more controllers, but only a single master controller can process new flow requests which is not for large networks in the current release. In addition, its management applications only operate on the controller(s) through fixed IP addresses that are assigned randomly without explicitly taking the load balancing needs into consideration.

Following ONIX [4], some important principles for the design and implementation of SDN controller cluster are described as follows:

A. Performance First

As the size of the network with SDN-Openflow enabled devices increases, the amount of new flow requests between the switches and the controllers grow accordingly. A scalable controller cluster is needed to ensure that the short new flow setup latencies are maintained for all switches in the network as the packet-in traffics from the switches to the centralized controller scale up. As such, the performance is always the most important consideration in our controller design.

B. Weak Consistency if Necessary

As the network scaling up, keeping strong consistency of network-wide states and related information will consume lots of computing and storage resources for the centralized multi-controllers in SDN. Similar to [4], in our approach, the controller cluster would trade some consistency for high performance, since most of the instantaneous inconsistencies would have negligible effects to the controller.

C. Light Weight Framework

The framework of the controller (cluster) should be as light weight as possible for large-scale centrally controlled networks, so that it can ensure quick response to all control traffics destined towards the controller cluster and achieve high performance at relatively low cost.

D. High Scalability:

In order to meet the growing control traffic demands for large-scale networks, the controller cluster should be able to scale seamlessly and autonomously, as long as the aggregated I/O capacities are not fully consumed.

E. Robust

A controller cluster should provide highest possible reliability, availability, and resilience to all kinds of potential network errors, including failures of devices, software, and more importantly malicious attacks to the centralized controller and its communication channel(s). Only with this stringent robustness, can it protect the network against significant loss to the business opportunities and avoid catastrophic consequences due to the possible single point of failure of the controller cluster. In our approach, the controller cluster with multiple servers runs in the equal mode, which effectively provides automatic fast failover and load-balancing.

III. Overview of the Proposed Centralized Controller Cluster Apporach

As discussed in the previous section, the new centralized SDN/Openflow control paradigm calls for robust and scalable controllers, especially for large networks with ever increasing and varying traffic volume. In this paper we propose an innovative multi-controller cluster and management mechanism to address the key issues in building a powerful and reliable centralized SDN-OF controller system that offers:

- Automatic load balancing and failover.
- Reduced communication overhead (synchronization and coordination between controllers).
- Scalability and extensibility.

The proposed cluster architecture is depicted in Figure 1, while [9] provides a detailed description on other core components such as the Network Information Base (NIB), Topology Management, Routing, Host Management, etc. The controller cluster consists of two major components: the message dispatcher (MD) or a MD Cluster (MDC), and the message processing cluster (MPC).

The main function of MD is to establish TCP channel connection from switches to a MD server node in the cluster in order to receive various packet-in(s) (PI) into some policy-based and prioritized input queues depending on the type of packets (for example, normal PI messages, status updates, or control messages, etc.) and their processing priority.

 ISBN: 978-1-61208-416-9

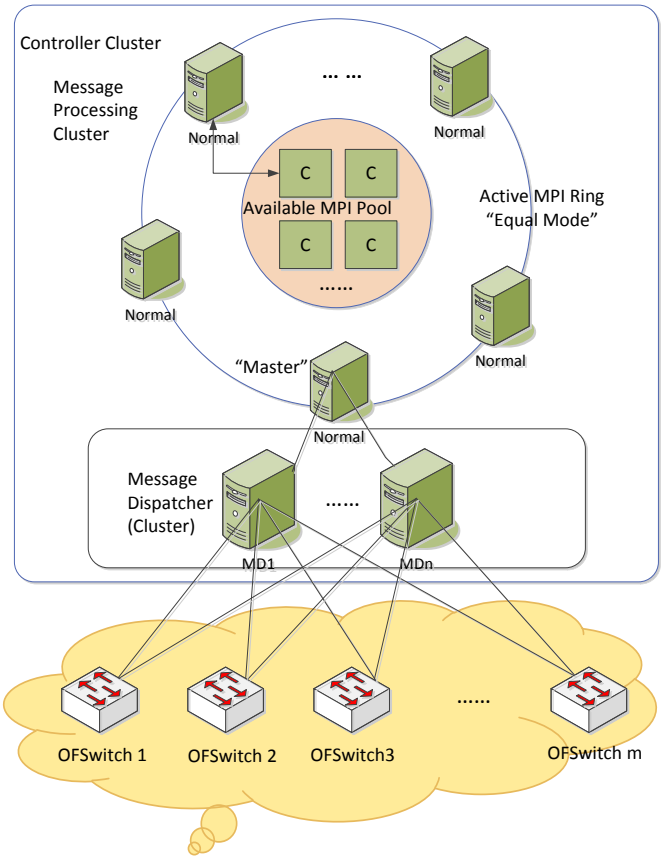

Figure 1. Diagram of controller cluster architecture

The MPC is responsible for processing the received packets, finding the proper path by invoking routing and resource allocation optimization algorithms, transforming the optimized routing decisions into flow table entries, and sending them back to the MD while the MD will eventually transport these flow entries back to all the switches en-route.

The following two subsections will discuss MD/MDC, and the MPC respectively.

A. Message Distribution Cluster (MDC)

A single message dispatcher server could control all switches and distributes all the messages to the MPC for a reasonably sized network. However its performance, especially its scalability could be limited by the total throughput of a single server, and also it lacks the required high availability and reliability for centrally managing a network of significant size, e.g. those networks with 1000 or more switching or routing nodes.

In our proposed controller cluster, we extend the single message dispatcher to a Message Distribution Cluster (MDC) with multiple dispatcher server nodes as illustrated in Figure 1. An OF1.3 enabled switch can be controlled by multiple controllers and support such features with a Message Distribution Cluster.

When the controller cluster is initiated, each switch is configured to establish a connection with all MDC servers. Then the MDC servers distribute the handshake messages received from the switches to the Message Processing Cluster MPC, while the "instantaneous MP master" (to be descried later) in MPC can record and maintain all the connected switches information after processing the handshake messages. In general, the MPC coordinates the management of the switches via consistent hashing to designate a MDC server for each switch. Policies can be configured such that each server within the MDC works in equal mode, while each server control (responsible for receiving, processing, and replying asynchronous messages) a number of designated switches. Currently, some rudimentary techniques are used for partitioning the network into "domains", mostly utilizing the network topology information, such as "neighboring" and hop-counts to measure how close a switch is to all other switches, and then grouping the "neighboring" nodes into the same domain. If the MP master detects some MD servers leave the cluster for any reason, it would reassign those switches originally controlled by those left MDC without overloading any remaining active servers in the cluster. As illustrated, the inherent ring structure of the active Message Dispatcher Instances (MDIs) and those available and ready to serve MDIs in the MDI pool effectively enable the use of the enhanced consistent hashing in order to make sure all the servers in MDC are utilized in a balanced fashion. Therefore the MDC could achieve the required high performance, adaptability, and scalability even when servers are added or removed from the cluster for any reason. The detail of the process is depicted in Figure2. For simplification, the internal active MDI ring and the available pool are not depicted herein.

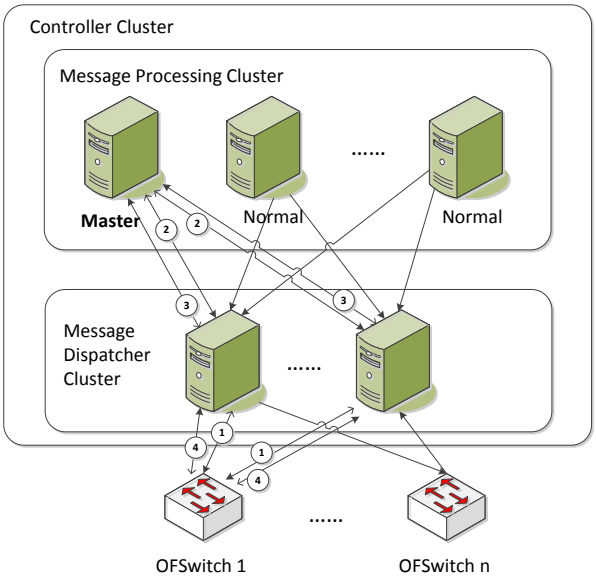

Figure 2. Managing Switches in Dispatcher Server(s)

The operation workflow has the following main steps:

- Step1: Switch(s) initially connects with all the dispatcher servers in the MDC.
- Step2: Dispatcher servers send the connected switch messages to the MPC, where the instantaneous "master" MPI could process them.

 ISBN: 978-1-61208-416-9

- Step3: The "master" MPI will decide which dispatcher server is assigned to control this switch.
- Step4: Dispatchers only reply to switches that they are responsible, in either master/slave or the equal mode.

B. Message Processing cluster（MPC）

The main function of the message processing cluster (MPC) is to receive and process switch messages from the MDC, send proper response messages (such as flow entries for routing the new packet-in) back to the MDC, while the MDC will dispatch those replies back to the relevant switch(s).

MPC can be a stand-alone server, or a cluster of physical servers on its own, each such server can provide multiple message processing instances (MPIs) at either the thread or process level [9]. Again for the sake of high performance and reliability, a specialized cluster architecture is used in our approach.

As depicted in Figure 1, the active MPIs within the MPC are organized into a ring structure, where each MPI would take the "master" role in turn for a short amount of time, while it will process the control and other status related messages within a certain specified time slot or until there are no more such messages to process, then it will proceed to become a normal MPI. For example, the "instantaneous master" is responsible for handling OF switching events, such as session connection, topology discovery, or status updates. Then it will quickly pull some packet-ins from the prioritized message queue and immediately pass the "master" role to the next MPI on the ring. After the previous "master" transitions itself into a regular "equal" mode MPI, it will start to process the packet-in message, and generate flow entries based on the routing and forwarding strategies and policies using the key packet-in attributes.

As network load changes, more or less MPIs may be needed, and if the traffic suddenly surges, the proposed approach should be able to respond instantly to add more MPIs. Therefore, a separate pool of MPIs are initialized, and updated with the latest information from the shared NIB, and any MPIs in the pool are ready to be pulled into the active ring to process packet-ins or other control messages. In [9, 10], more details are presented on how the NIB, and the multiple priority queues are architected in the controller cluster. In this way, MPC can achieve high availability and automated load balancing with the desired high performance. In addition, it can significantly reduce the management work load, because all MPIs are identical in structure and can be cloned from a uniform infrastructure.

C. Packet-In Message Processing

We categorize messages handled by the MPC into two types: The control messages that cover all Openflow protocol messages, along with various events originated from network status or policy changes, and the normal packet-in (PI) messages that include forwarding-request packet (data flow, host-to-host request data flow etc.).

1) Control Messages

The control message is mainly used in the control layer, and they also provide critical information needed for the data forwarding layer (normal PI messages). As depicted in Figure 4, only the "master" controller will process those control messages, therefore the cost to achieve much desired consistency across all the MPIs in the cluster is minimized.

Typical control messages include：

- Protocol interactive messages (handshake messages, port status, etc.) between controller and switches.
- Some special control messages wrapped in PI Messages, such as LLDP link discovery messages.
- Static messages (basic configuration, policy configuration, etc.).
- Events due to network status changes.
- Messages or other data generated dynamically through the northbound APIs.

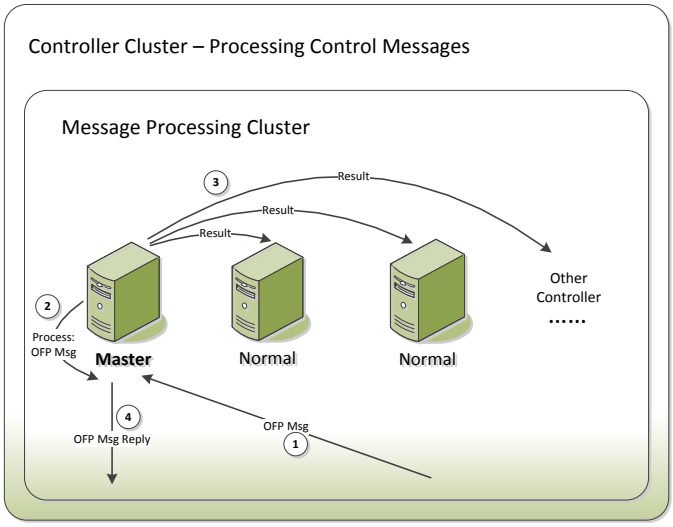

Figure 3. Processing Control Messages

Some of the control messages have strong dependencies with each other, and the order of processing them become critical to maintain the proper consistency. This is one of the main reasons why the current instantaneous "master" approach is taken, in order to process all control messages with one MPI at any given time. Therefore, not only it can enforce the proper processing order, but it also makes the updates to the other MPIs and the shared central NIB consistent at minimum costs. In case that the "master" MPI becomes dysfunctional, the next MPI on the ring would take over the "master" role. When this occurs, some control messages or their processing results may get lost, that could result in some temporary inconsistency in the network. However, the enforced short maximum time interval to transfer the "master" role to the next available MPI would limit the consequence of the inconsistency to an acceptable level. Furthermore, after the controller completes topology discovery and feature learning, there will be much more reading operations than writing in a SDN network, while only the failed writing operation could lead to such

 ISBN: 978-1-61208-416-9

inconsistency. The proposed approach properly follows what is recommended in [4], and innovatively provided a solution that greatly improves the processing performance with affordable costs, while allowing certain acceptable weak consistencies at certain duration-controlled time intervals.

2) Equal Mode for Normal Packet-In Messages

For normal PI messages, the equal mode is adopted, in which all servers (threads or processes) share the processing loads while providing backups to each other. As discussed earlier, the "master" MPI will pull a packet-in message from the normal PI message queues, transfer the control to the next MPI on the ring, and then start to process the new packet-in, as all other active MPIs will do.

Controllers in SDN are to process new flow routing requests by collaborating with upper layer management applications, such as routing and resource allocation, transforming the resulting decisions into flow entries, and then distributing them to all OF-switches along the designated path(s).

In the proposed approach, each MPI uses the local cached data, fetched or synchronized from the central NIB to perform the above tasks.

IV. Adaptive Resource Control Model in MPC

In order to maximize the performance of the contemporary powerful multi-core CPUs and available I/O throughput, an adaptive resource control model is established. Today's abundant mobile internet traffics are dynamic and bursty, complicated with different QoS requirements. They necessitate finer-grained control with enough pre-provisioned resources to make sure that the perceived performance is acceptable even in worst situations. The providers, however, need to properly control the use of those resources for their increased utilization, and at the same time, it is aimed to reduce the network management complexity and operating cost. For example, if it can effectively apply SDN based centralized control with adaptive and optimized routing and resource allocation [12], they can offer to dynamically reroute the traffics, enable or disable the use of certain routes, and increase or decrease capacities on some routes. As traffic patterns in general would be quite different between working hours and off hours, such dynamic routing decisions would not only help achieve their business objectives, but also enable a greener computing and data networking paradigm.

With the proposed approach, as the new packet-in requests varies, the number of concurrent threads in each MPI, and also the number of servers allocated to the MPC can be increased or decreased dynamically. However, after numerous experiments and comparison studies, it was recognized that increasing the number of threads or servers may not necessarily improve the actual single MPI or the entire MPC processing capability, especially when we have to deal with a large number of PIs from a large scale network. When the number of concurrent threads in a MPI or servers in MPC increases beyond a certain limit, the newly added thread or server can consume additional precious data bandwidth, CPU processing power, and memory resources, leading to the increased risks of data locking, exceeding the limit of the total available I/O capacity, and eventually, the overall performance degradation.

The use of the Multi-dimensional DHT (MDDHT) [11], along with the MDC and MPC in the proposed approach, can learn different traffic patterns and the related parameters to discover the hidden relationships between the traffic loads and the proper number of MPIs/servers needed for efficient network traffic control under various operating conditions. Such parameters include the number of packet-in messages that a thread or a server can process in a second, given its CPU/memory processing capabilities, the I/O capacity, and the bandwidth at the port and the aggregated switch level, etc.

V. Performance and Scalability Consideration

Our main objective is to design and implement a powerful controller cluster to address the critical performance issue for the centralized controllers in a large SDN-OF network, while providing the desired fault tolerance and load balancing capabilities. As discussed before, slightly sacrifice made in the consistency with some acceptable packet losses to exchange for significant gains in performance [4], should be an excellent overall trade-off. In addition, such a compromise would render the proposed approach much more cost-effective.

In an earlier design of the proposed approach, direct connections, through some centralized and prioritized packet queues, were established between the switch and the MPC, without a dedicated MD or MDC. But soon we found that such a "centralized" and "single I/O" mechanism would limit the throughput of the controller/MP cluster and become a bottleneck, causing the total throughput of the cluster to decrease and the latency of processing each individual request to increase, no matter how powerful the MPC is.

Later MDC was introduced between the controllers/MPC and switches. Even though the direct connection model could provide faster response time for new flow requests if the overall I/O limit is not reached, but it could be degraded at an explosive rate soon after the sustained I/O is approaching the system upper limit. With multiple servers in the MDC, where each server brings in its own portion of I/O capability, the overall throughput of the proposed approach can be improved almost linearly until reaching the limit imposed by the capability of the MPC. On the other hand, it is found that accelerating TCP/IP communication performance by DPDK technology [13] and high-performance TCP/IP protocol stack can significantly improve the performance of both the MDC and the MPC.

VI. Scalability and Performance Analysis

In this section, we review how we conducted performance testing experiments and present related comparison results. We focus on how the proposed approach could improve the performance, scalability, and reliability. In particular, we study how fast such a centralized controller cluster could process new PI messages

 ISBN: 978-1-61208-416-9

in a large-scale communication network, and investigate on how certain known issues could be addressed to avoid potential performance bottleneck.

A. Experimental Methodology and System Setup

We designed and implemented the proposed scalable centralized controller cluster prototype with one Message Dispatcher Instance with one MD server and several Message Processing Instances (MPIs) running on two to four servers, and we compared the throughputs achieved by such a cluster with different number of MP servers. For a fair comparison, we first evaluated our centralized controller cluster with only one MPI in the MPC, and after that, we moved on to evaluate the throughput of the cluster with multiple MPIs in MPC.

As shown in Figure 6, we constructed a network topology with two OpenFlow switches, one Message Dispatcher Instance (MDI), and several Message Processing Instances (MPI). They were run on separate 2.4GHz 64bit Intel Xeon E5620 server machines with 8GB RAM to avoid interferences, where all data links were 10Gbps. It is also important to note that for each processed packet-in (PI) message, the centralized controller cluster would reply with one packet-out (PO) message and two flow-mod (FM) messages. In general, N separate FM messages would need to be created and distributed if there are N nodes en-route for a PI. Therefore, in our experimental configuration, there were three reply messages for each PI with a 1:3 (or N+1 in general) ratio between the receiving and replying endpoints in the centralized controller cluster.

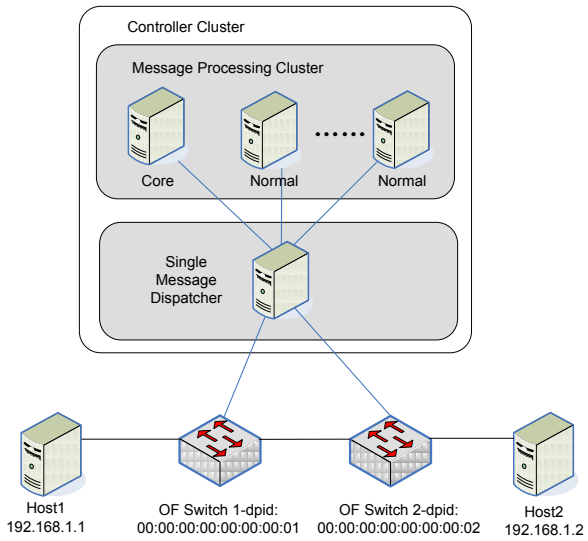

Figure 6. Experiment setup

To our knowledge, currently published controller performance results are based mainly on using only ONE switch. Consequently, the ratio of the number of PI messages and the number of PO plus FM messages are only 1/2. In addition, the controllers and switches are mostly only OF 1.0 compliant, supporting only one flow table. In our experiments, the OF switches were all compliant to the newer and more sophisticated OF 1.3 standard, and the controller and the switches actually supported a flow pipeline with nine flow tables optimized for typical network applications, such as firewall, ACL, layer 2 or layer 3 transporting, VLAN, etc. Therefore, what being tested and presented here are more practical as the older version of OF 1.0 cannot support many critical network functions for existing data services.

We injected traffic flows from one host to another based on a fixed but configurable rate. Each flow was generated as a single 64 Byte UDP packet. Consequently, in our experiments, each PI message was effectively 84 Bytes, while PO messages were 80 Bytes each, and FM messages were 136 Bytes each respectively.

B. Experiments with one MD and 1-5 MPIs in MPC

We computed the rate of packet-out (PO) and flow-mod (FM) messages that would be created and replied from the controller cluster. As depicted in Figure 7, we initially studied and compared the centralized controller cluster's scalability and performance with varying number (1 to 5) of MPIs.

Message Processing Instances	Packet-in		Packet-out		Flow-mod		Packet-out + Flow-mod	
	Packet Rate (kpps)	Packet Bandwidth (Mbps)	Packet Rate (kpps)	Packet Bandwidth (Mbps)	Packet Rate (kpps)	Packet Bandwidth (Mbps)	Packet Rate (kpps)	Packet Bandwidth (Mbps)
1 Instance	560	376	590	358	1120	1200	1680	1558
2 Instances	1120	752	1150	716	2240	2400	3360	3116
3 Instances	1660	1115	1690	1062	3320	3600	4980	4662
4 Instances	2240	1182	1760	1126	3520	3800	5280	4926
5 Instances	2800	1182	1760	1126	3520	3800	5280	4926

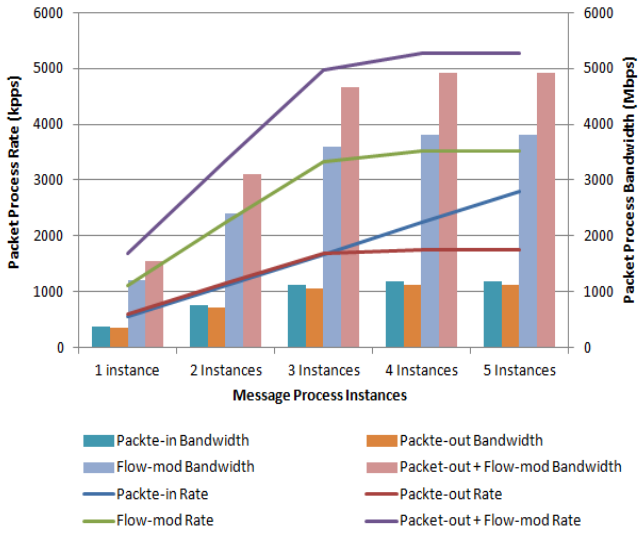

Figure 7. Centralized Controller Cluster throughput Vs. the number of Message Processing Instances: 1 MD Server

As can be seen from Figure 7, when the number of MPIs was three or less, the throughput of the centralized controller cluster could scale linearly with the number of MPIs, where each MPI could process 560K PI messages per second. But as we continued to increase the number of MPIs, the throughput of the centralized controller cluster stopped increasing linearly and eventually stagnated.

Further drill-down analysis revealed that with four MPIs, the centralized controller cluster could process 1.7 million PI messages per second; as a result, 1.7 million PO messages and 3.4 million FM messages would need to be generated and replied to the 2 switches in the experimental network per second. As such, the MDI would consume about (1.7 million POs * 80 Bytes/PO * 8 Bits/Byte + 3.4 million FMs * 136 Bytes/FM * 8 Bits/Byte), or requiring about a total of 4.8 Gbps bandwidth in one direction. That has almost exceeded the maximum TCP replying capability using the standard Linux socket API.

As the number of Message Processing Instances continues to increase, I/O performance eventually become the main bottleneck that impacts the scalability and the throughput of the centralized controller cluster.

C. *Effects of Increasing the number of MDs in MDC*

In order to overcome the I/O limit imposed by one MD server in the above experiment and fully utilize the processing capabilities of the MPC, two MD servers were used in a new experiment. Each MD server was assigned to be responsible for one switch just to simplify the matter. In practice, such as in the distributed SOX [10], some mechanism is needed to partition a large network so that each MD server could be assigned to be responsible for a subnet.

Figure 8 presented the result. As it can be seen, the total throughput now scales linearly as the number of MPIs increases. With the proposed approach, the MPC can be scaled easily to increase its processing capacity with additional MPIs in MPC, and now with the additional MD servers in the MDC to overcome the I/O limit, the centralized controller cluster can manage much larger network with adequate I/O bandwidth such that the overall system can scale up linearly.

Message	Packet-in		Packet-out		Flow-mod		Packet-out+ Flow-mod	
Processing Instances	Packet Rate (kbps)	Packet Bandwidth (Mbps)	Packet Rate (kbps)	Packet Bandwidth (Mbps)	Packet Rate (kbps)	Packet Bandwidth (Mbps)	Packet Rate (kbps)	Packet Bandwidth (Mbps)
1 instance	560	376	590	358	1120	1200	1680	1558
2 instance	1120	752	1150	716	2240	2400	3360	3116
3 instance	1680	1128	1740	1074	3360	3600	5040	4674
4 instance	2240	1504	2330	1432	4480	4800	6720	6232
5 instance	2800	1880	2920	1790	5600	6000	8400	7790

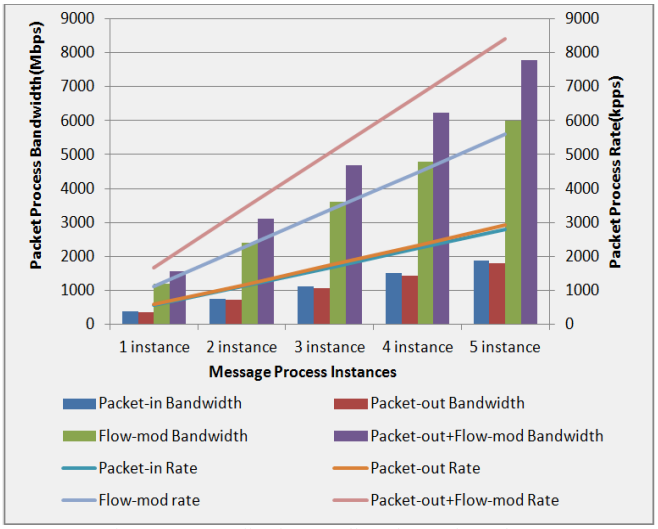

Figure 8. Centralized Controller Cluster throughput Vs. the number of Message Processing Instances (2 MD Servers)

D. *Other Considerations*

Furthermore, with this centralized controller cluster, one MD instance/server can be enough to match the PI message processing capacity of the deployed MPC. If more PI messages need to be processed, distributed controller clusters should be used, while each cluster manages a smaller domain or subnet of the large scale network with enough I/O bandwidth between the controller cluster and all switches it controls as in DSOX [10].

As higher speed (such as 40 Gbps or even 100 Gbps) NIC cards become available, obviously both processing power of MDC and MPC can be significantly increased. As it is relatively easy to increase the number of servers in the MPC to achieve much higher PI message processing rate, more MDIs (or servers) in the MDC should be added to further balance the load of the much increased traffics between the centralized controller cluster and the switches.

 ISBN: 978-1-61208-416-9

VII. CONCLUSION AND FUTURE DIRECTIONS

This paper presented a scalable centralized controller clustering approach for SDN with Openflow. In our approach, the controller instances (threads, and processes on a single or multiple physical servers) would handle the control and normal data packets differently, in which the packet-in message processing was carried out by different controller instances in the cluster running in equal mode. It is aimed to achieve maximum performance with slightly sacrificed but acceptable and temporary inconsistency. The controller cluster in our approach demonstrated excellent scalability until the system I/O limit was reached. A self-learning and adapting mechanism was also incorporated in our approach to further optimize the number of controller instances under varying operating conditions to adequately handle frequent changes in network states and traffic flows. In addition, the described approach provides automatic load balancing and fast fail-over at almost at no extra cost.

Experimental studies were conducted, and the test results with varying number of physical servers (or processing instances) were presented and studied. It clearly demonstrated the feasibility and advantages of the proposed approach.

Efficient control and management of large scale networks based on SDN is an active research area. Many issues remain to be investigated, such as the balance between the proper level of consistency and the high performance in a distributed cluster environment, the interoperability of SDN with existing IP networks, dynamic network resource utilization, etc.

REFERENCES

[1] Open Networking Foundation. Software-Defined networking: The new norm for networks. ONF White Paper, 2012.

[2] "OpenFlow Switch Specification 1.3.0", Open Networking foundation.

[3] A Tootoonchian, Y Ganjali, "HyperFlow -- A Distributed Control Plane for OpenFlow", Proceeding of the 2010 INM conference, https://www.usenix.org/legacy/event/inmwren10/tech/full_papers/Tootoonchian.pdf

[4] 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", Proceedings of the 9th USENIX OSDI conference, Vancouver, 2010. http://static.usenix.org/events/osdi10/tech/full_papers/Koponen.pdf

[5] A Dixit, F Hao, S Mukherjee, T Lakshman, R Kompella, "Towards an Elastic Distributed SDN Controller". Proc. of the ACM SIGCOMM Workshop on HotSDN. pp. 7-12, Hong Kong, August 2013.

[6] S Yeganeh, Y Ganjali. "Kandoo: a framework for efficient and scalable offloading of control applications", Proc. of the ACM SIGCOMM Workshop on HotSDN, pp. 19-24,_,Helsinki,Finland, August 2012

[7] M Yu, J Rexford, M Freedman, J Wang, "Scalable flow-based networking with DIFANE", Proc. of the ACM SIGCOMM. pp. 351-362, 2010

[8] OpenDaylight, http://www.opendaylight.org/, [accessed: 2014-12-18]

[9] M. Luo, Y Tian, Q Li, J Wang, W Chou. "SOX –A Generalized and Extensible Smart Network Openflow Controller(X)", The First SDN World Summit, Germany, October 2012. http://www.layer123.com/download&doc=Huawei-SOX_WP_V1.0

[10] Q.Li, K. Lin, M. Luo, etc., "DSOX: Architecture and Design," Technical Report, Huawei Shannon Lab, May 2014.

[11] M. Luo, X. Wu , Y. Zeng, J. Li, K. Lin, B. Man, and W. Chou, "Multi-dimensional In-Memory Distributed Hashing Mechanism for Fast Network Information Processing in SDN", accepted by the 9th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS-2015), Blumenau, Brazil, July 2015

[12] M. Luo, Y. Zeng, J. Li, W. Chou, "*An Adaptive Multi-path Computation Framework for Centrally Controlled Networks*", accepted by Journal of Computer Networks, Elsevier, February 2015

[13] Intel® DPDK: Data Plane Development Kit. http://dpdk.org/, [accessed: 2014-12-18]

 ISBN: 978-1-61208-416-9