Performance Comparison of a SDN Network between Cloud-Based and Locally Hosted SDN Controllers

Kashinath Basu[1], Muhammad Younas[1], Andy Wan Wac Tow[1] and Frank Ball[2]

[1]*School of Engineering, Computing and Maths*
Oxford Brookes University, Oxford, UK
{kbasu, m.younas@brookes.ac.uk}
[2]*Frank Ball Consulting,*
Oxford, UK

***Abstract*-- In a SDN network model, the robustness, scalability and reliability requirement of the control plane makes it an ideal candidate for being hosted on a cloud infrastructure. In addition, the control plane performs large volume of data processing from packet headers to network monitoring data in order to provide adequate level of QoS to the traffic. The realization of a cloud based SDN networking approach is predominantly dependent on the performance of the SDN controllers on the cloud environment. This paper presents a comparative study of the performance of a SDN network between a locally hosted SDN controller within the enterprise with a cloud based remote controller. Since a wide range of SDN controllers are available in the market with different levels of functionalities, performance and complexities, the analysis is validated by comparing the results across three different types of controllers. Furthermore, the impact of the network topology on the performance of the controllers is further validated by comparing the performance across two different topologies. In addition, a comparative performance analysis of the throughput and a theoretical evaluation of the controllers are also presented.**

Keywords — Bigdata, Cloud, FloodLight, Mininet, NOX, Open vSwitch, Performance analysis, POX, SDN controller.

I. Introduction

Traditional networking has a number of features which provided the robustness in the early days, but also acted as a hindrance to the progress. For example, in data forwarding, the path selection process is typically distributed and made by the forwarding devices of a network comprising of switches and routers which mainly had local or regional knowledge of the topology without the complete understanding of the overall network. Although this made the network more fault tolerant as even after a failure on one segment still allowed the network to operate in the remaining segments, it however prevented a centralized understanding of the overall network condition and thereby restricted network wide end-to-end intelligent monitoring or policy level decision making. For traditional data requiring best effort forwarding this was adequate. However, present day networks carry a wide variety of application data comprising of audio, video, text, etc. with realtime, streaming, non-realtime and interactive delay requirements. Providing end-to-end Quality of Service (QoS) to these complex range of traffic can be simplified by centralized QoS provisioning and orchestration which is difficult with the traditional network setup. In addition, the hardware of the current networking devices are limited in terms of the volume and scale of data they can handle. This has not been an issue where the scope of processing has been restricted to only local data, but this is inadequate for handling the entire network level processing.

Historically, the evolution in the field of networking has also suffered due to the closed nature of vendor devices which has restricted flexibility and management of complex networks as well as hindered research and progress in the field. In this context, the Software Defined Network (SDN) [1] approach has unleashed the opportunities of accelerated innovation and development in the field by decoupling the data forwarding ASIC (Application-Specific Integrated Circuit) from the control and logic to divide the network into three overlay planes. The topmost northbound application plane is responsible for policy level decisions which are compiled into flow rules by the mid-level controller plane. In some modest controller such as POX for example, these application level policies can be written as libraries of the controllers itself, whereas in more complex controller framework such as Ryu [2] they run as separate processes and communicate with the controller using JSON and REST APIs. The role of the controller is that of a network operating system. The flow rules generated by the controllers are pushed to the forwarding devices in the southbound infrastructure plane (Fig. 1).

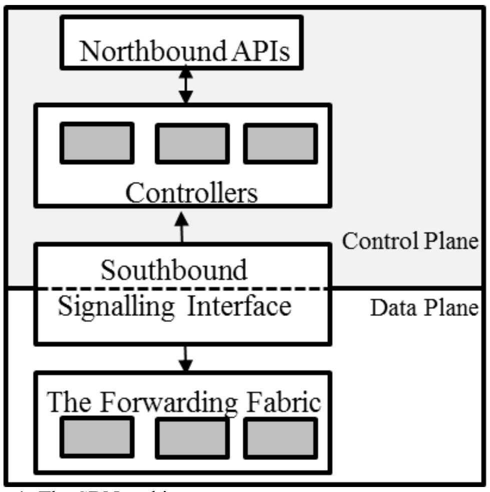

Fig. 1: The SDN architecture

An enterprise scale SDN controller typically has several components. Some of these components are north and south bound drivers, a range of flow rule tables, forwarding flow tables, performance monitors, tracers/logs, etc. In a multi-controller environment for a large scalable network there are additional modules to manage synchronization between the controllers, load balancing, virtualization aspects, etc. All these components work in synch to provide the overall performance and functionalities of the controller [3]. This requires storage and processing of large volume of big data.

With the data forwarding delegated to the infrastructure layer, the controller and the northbound APIs are solely responsible for network wide software logic and the data processing activity. This makes the controller an ideal candidate to be hosted on a cloud environment to leverage on the scalability, reliability, security, compute and storage facilities offered in a cloud. Additional value added services such as specialized firewalls, QoS broker, etc. can be further implemented since the constraints on resources are removed on a cloud implementation. This architecture facilitates a cloud based SDN network with the infrastructure layer hosted in the physical network and controller and associated APIs hosted on a cloud.

In an enterprise environment, however the performance especially in terms of access delay and throughput of the remotely cloud based controller should be at least identical to that of a locally hosted controller operating in a lightly loaded condition. By default, it can be assumed by the very nature of cloud computing that a SDN network with a cloud based controller will exceed the performance benchmark of that of a locally hosted controller in a heavily loaded network condition due to the constraints of local resources; however the real test lies in providing at least an identical level of service in lightly loaded traffic condition.

There can be other factors such as load, topology and the characteristic of the controller which may impact the performance of the network. For example, there are a wide range of SDN controllers with different functionalities and features suitable for different scenarios. In order to address these variabilities, the analysis here considers the performance with different traffic load, network topologies and categories of SDN controllers. The objective here is to identify the general trend in overall performance of a cloud based SDN control plane to that of a locally hosted one. In the context of the controller, the paper analyses the performance using three different well-known SDN controllers of three different varieties viz. NOX popular for its suitability for rapid prototyping, POX for speed and performance and Floodlight for its scalability and the extensive set of features. The performances of the controllers are compared in terms of throughput and delay. While evaluating the performance, the components of the controllers are also individually investigated and as a byproduct of this work a comparative study of the controllers has also been presented.

The paper is organized as follows: Section II presents some of the key features and advantages of cloud computing; Section III gives a background to the softwarization of network leading to SDN and discusses some of SDN's key features relevant for remote cloud based hosting. This includes flow table compilation for the switches and the signaling between the controller and the switches. Section IV presents the configuration and setup of the two sets of experiments; The results and analysis are presented in section V; Finally, section VI evaluates the contribution and identifies future work.

II. Cloud computing

Cloud computing provides a platform in order to deliver computing services and resources over the Internet such as compute power, storage, servers, databases, networking, and software applications [4]. The use of cloud computing has significantly increased over the recent years. The main driving forces is that cloud computing provides flexible, scalable and on-demand IT services to small, medium and large scale businesses organizations. With the popularity of cloud computing, a large number of companies such as Amazon, Google, Salesforce, Microsoft, IBM, and many others are offering cloud services to cloud service consumers.

For the service consumers, cloud provides various benefits [5, 6]. First, it is more economical for service consumers (businesses and organizations) to host their IT services in the cloud than hosting them locally. Cloud eliminates costs such as buying and maintaining local IT infrastructure of hardware and software. Second, cloud provides scalability and elasticity where resources are provided on-demand when needed. Third, cloud computing provide better reliability as data is mirrored (replicated) across multiple sites of cloud vendors network. In addition, cloud provides better security, performance and flexibility.

Cloud services are provided using different service provisioning models. These include, for example [7]:

Software-as-a-Service (SaaS): In SaaS, software applications are hosted in cloud computing. Users can use them using Web and Internet. For example, web browsers (or dedicated APIs) are used to use SaaS services. There are various companies that provide SaaS services. Common examples are Google Apps and Salesforce.

Platform-as-a-Service (PaaS): In this model, service consumers can use cloud services in order to create and run applications. The underlying infrastructure is not managed by service consumers. Instead they use that in order to build, create and run their applications. Examples of PaaS are Google App Engine. Heroku, Windows Azure.

Infrastructure-as-a-Service (IaaS): IaaS provides service consumers with hardware resources such as memory, CPU, and storage space in order to deploy and run their software applications. In this model, service consumers can have control over the underlying cloud resources such as operating systems, storage, and their applications. IaaS examples include, Google Compute Engine, Rackspace, and Amazon EC2.

Cloud based SDN control plane lies in the category overlapping the PaaS and IaaS model. Here the controller acts as a SDN platform in the cloud. Either the compute resources such as CPU, memory, etc. can be allocated explicitly like the IaaS or can be provisioned dynamically by the cloud based on the processing load on the controller in the control plane like the PaaS model.

III. BACKGROUND AND KEY FEATURES OF CLOUD BASED SDN

The concepts of centralization, programmability and virtualization in networks are not recent. For instance, at the beginning of the 80s AT&T introduced the concept of "Network Control Point" (NCP). Historically the same channel was used to carry data and call control signaling information. With NCP, all the mandatory signaling process for call management was centralized inside the NCP resulting in more secure and clear separation and management of control and data. Later in the 90s the concept of programmable networks called "Active Networks" was proposed. It allowed implementation of different types of services in network devices such as firewall or DNS service on a router or switch (Feamster, 2014). In principle, this is similar to the services hosted as Northbound APIs on a controller in an SDN network, but centralization was not part of Active Network's strategy and therefore these services were not scalable and restricted to limitations of the hardware of the forwarding devices.

However, with SDN the control plane is centralized, decoupled from the bare metal hardware centric data plane and designed as a software centric layer. In principle, this makes the SDN controller ideal for Platform as a Service (PaaS) or Infrastructure as a Service (IaaS) over a cloud. However, in order to provide the seamless service remotely, there are few architectural issues that need to be carefully considered since there is network communication involved between the switches located in the infrastructure layer and the controller in a remotely hosted cloud.

In an SDN network the notion of flows is crucial. Packets from the same source-destination application pair are grouped as a flow. The communication between the controller and switch to resolve the route of these flows can be broadly based on two approaches: in the "reactive" mode, when the first packet from a flow arrives at a SDN switch, the packet is pushed to the controller since the forwarding table will initially not have any stored route entry for the packet. The controller based on its forwarding policy will process the route for the packet and push a route entry in the flow table of the switch. Subsequent packets from the flow can be forwarded directly from the switch without consulting the controller thereby reducing the additional processing delay at the controller level. The flow entries are also timestamped and hence if no new packet arrives before this timeout (soft timeout), the entries are purged and the entire process has to be repeated. In contrast, setting longer timeout period may reduce switch-controller communication and processing but introduce outdated stale entries in the flow table. Switches are also configured with hard timeout after which an entry is deleted irrespective of the last refresh interval.

Alternatively, the switch-controller communication can be in "proactive" mode where static flow rules are pushed in advance based on forwarding policies before the arrival of the flow thereby reducing controller-flow communication and flow level latency. This however requires more complex in-advance management of the flow policies.

These optimizations can reduce the realtime on demand switch-controller communication which could be a bottleneck for a cloud based remotely hosted controller. This communication is signaling intensive and dependent on the signaling protocol used. There are a number of alternative signaling protocols that could be used viz. OpenFlow, ForCES, I2RS, GMPLS, NetConf, PCE etc. [8, 9, 10]. However, OpenFlow [8] is the most common among them and it has a rich set of signaling primitives and is widely supported by vendor hardware devices and controllers. This work is based on the OpenFlow protocol for communication between the switches and the SDN controllers.

IV. EXPERIMENT CONFIGURATION AND SETUP

From the large number of SDN controllers in the market, three different controllers were selected each of which is dominant in its own category. For research and rapid prototyping purpose, POX is widely used and the "dart" branch of POX [11] was shortlisted here for this category. It is written in Python and has large community of researchers and freelancer and wide range of open source libraries. The second controller shortlisted was NOX [12]. It is C++ based and has a high-level programmatic interface for C++ and Python. It is known to be relatively fast in terms of performance. The final controller for the analysis was FloodLight [13]. It is Java based industrial scale controller with a large volume of libraries to support various types of enterprise level functionalities. All these three controllers use OpenFlow as the signaling protocol for the southbound interface between the controller-switch. Using a single type of signaling interface provides a uniform experiment base and eliminates any relative performance advantage of one controller over another.

At the top level, two sets of experiments were planned: experiment set (A) to measure the relative latencies between the three topologies using the different controllers and experiment set (B) to measure the relative throughput of the controllers. The former set of experiments assists us to understand the impact of a cloud based controller compared to a locally hosted one. The later experiment provides us with a holistic comparative overview of these three categories of controllers. The performance of an individual controller can vary significantly based on several external and internal conditions such as hosting hardware specification, network topology, volume and nature of traffic, etc. Therefore, here in each of the experiments, we compared the performance under identical conditions. The network topology consisted of hosts connected to Open vSwitches [14] and was hosted on a Mininet [15] emulator running on VMware hypervisor. The virtual machine (VM) was configured with 8GB memory and 4 virtual cores, each running 2.3 GHz. The remote controller was hosted on an external VM with 16GB memory and 8 2.3 GHz cores.

A. Experiment Set(A): Setup for the Delay Monitoring Experiments

The experiments were run separately with three different network topologies and three different controllers to identify the general overall trend and negate the effect of topology and controller specific performance issues.

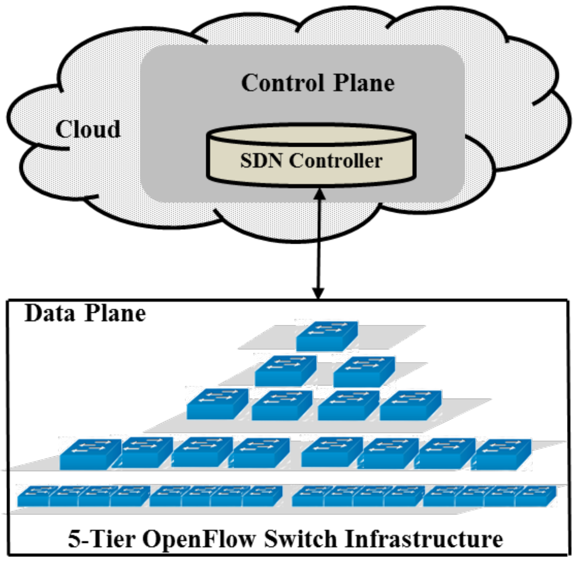

Fig. 2: Architecture of the cloud based SDN network

In topology 1, 32 OpenFlow switches were arranged linearly, each connected to the next one and also to a locally hosted controller. Also, one host is connected per switch. Topology 2 consisted of a hierarchical tree-based architecture with five layers and a fan-out factor of "2" per node using 31 OpenFlow switches and 32 hosts. The switches are connected to a local controller. Topology 3 has a similar network setup but with a remotely cloud based controller (Fig. 2). All local links were configured with a bandwidth of 1 Gbps and 0.1msec delay. The link level loss was set to 0% to avoid any extra latency due to packet loss. In topology 3, the remote controller was connected to the network with a 20 mbps link with background traffic.

The delay characteristics of the controllers were compared by monitoring the round trip time of the end-to-end ping delay across the network topologies with different controllers. The pingFull method of the Mininet class (net.py file) was used to generate the ping messages and to capture the performance statistics directly from the virtual hosts. 15 iterations of the ping messages were sent from each host to get a holistic overall performance estimation.

Each ping message creates two flow states in the switch for a ping-request and a ping-reply flow. In addition, the first ping message from a host also creates two additional states for an ARP-request and an ARP-reply message. This also has an additional round rip latency impact for the first ping message. In our experiments, the results from the first iteration of the ping messages from the hosts were discarded to avoid this initial ARP exchanges. It is also important to note, that once a flow entry is created for a particular source destination pair, it will stay in the switch till either a hard or soft timeout occurs as mentioned in section III. Any new ping messages in-between will not trigger any new packet-in or corresponding flow-mod message to or from the controller respectively. Hence, all controllers' hard timeout period were recompiled to two seconds in order to flush every flow after a short delay and the retransmission time of the ping messages were configured accordingly with a sleep interval of 5 seconds to avoid retransmission within that period. This will ensure that all pings trigger a packet-in message so that we can distinctly compare the impact of the switch-controller communication in the performance between the topologies.

B. Experiment Set(B): Setup for the Throughput Monitoring Experiments

The suitability of the topologies to handle enterprise scale load was tested by a comparative analysis of their throughputs. The Cbench [16] tool from the OFlops testing platform was used for this purpose. The role of Cbench was to send multiple "packet-in" events towards the controller, in order to simulate the need for a flow. Then, by monitoring the "flow-mod" messages replied by the controller, the raw throughput of a controller to compute flows can be estimated. The configuration of Cbench was set to simulate 32 switches and 100K MAC addresses per switch. 20 iterations of the test were run each lasting 10 seconds. Cbench was also set to operate in "throughput mode" which means that the simulated switches were allowed to send as many requests as their buffer and virtual resource permits.

In addition, certain changes were made to POX and FloodLight controllers to facilitate Cbench to operate smoothly. In POX's default switch module, it appeared that the controller was systematically ordering to flood traffic through packet-out messages when using Cbench. Therefore, no "flow-mod" were pushed. To avoid this issue, the flooding algorithm was rewritten to push flows for more accurate results. In FloodLight a throttling mechanism is implemented to restrain any heavy suspicious request flows. This feature would have conflicted with Cbench's intended operation. To avoid this issue, the throttling mechanism was deactivated and the controller was recompiled.

V. RESULTS AND ANALYSIS

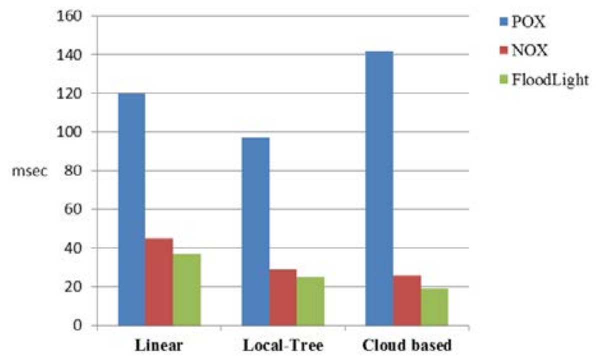

Figure 3. Round trip end to end delay.

Fig. 3 shows the mean round-trip end-to-end delay of the three controllers using the three different topologies discussed in section IV(A) and running the pingFull method. For all of the controllers, it can be seen that the topology has a direct effect on the performance of the controllers. As expected, the performance of a structured hierarchical tree based topology has been better than a linear flat based one. Within the two hierarchical topologies, the cloud based controller's performance has been better than the locally hosted controller in the case of NOX and FloodLight. This demonstrates that any extra latency in the switch-controller communication path in the cloud based controller case is compensated by faster processing time of the packet-in messages due to the additional computing resources in the cloud.

Among the controllers, the values show that in general the delay for POX is significantly higher than NOX and Floodlight across the three topologies. It is also seen that the percentage difference in delay for POX between the linear and the local tree topology is comparatively less compared to the other two controllers. On investigating the architecture of POX and Open vSwitch, it was found that POX generates one instance per virtual switch [17]. This therefore to certain extent keeps the POX overhead scalable and transparent to the topology of the network. The third experiment in this set using the cloud based controller however shows that POX is less robust in dealing with link condition as the delay in this case is significantly higher.

The delay figures represented in Fig. 3 includes along with the switch-controller interaction related delays additional factors such as transmission, propagation and queueing delay. In order to eliminate the impact of those additional delay components, the experiments in set (A) were rerun with static flow tables and plugging off the controllers. This then captured the delays only associated with the transmission, propagation and queueing. Following this, these values were subtracted from the corresponding all-inclusive delay values to derive the effective switch-controller associated processing delay (Fig. 4). The delays represented in figure 4 is comprised of the following four factors: 1) The time taken for the Open vSwitch to compute a "packet-in" message for the controller; b) the controller's processing time to parse the "packet-in" message and compute a proper flow from this message; c) the time for the controller to push this flow through a "flow-mod" message to the switch; d) the time taken for the switch to parse the "flow-mod", install the according flow, "un-buffer" the awaiting packet and send it through the right output port specified by the "flow-mod" message; e) in addition, for the cloud based controller, it also includes switch-controller round trip transmission delay.

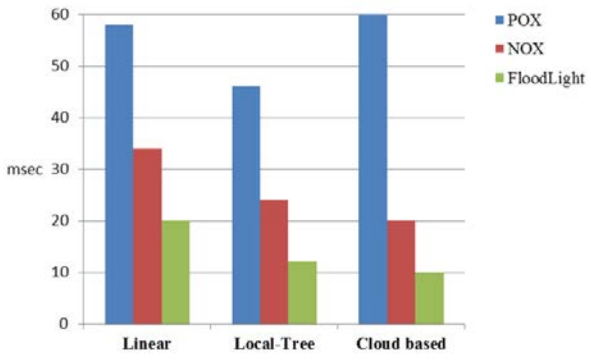

Figure 4. Controller-switch communication and processing related delay.

Although the delay values are much smaller here across the three topologies as expected since only the controller-switch associated delays are presented, but the same trend in the comparative performance of the three controllers are visible in all the three scenarios. In the first two topologies, the delay of any particular controller is different in the two cases mainly due to the difference in the number of times a controller is solicited (due to the difference in topology) and congestion in the controller-switch path. However, comparing the cloud based with the local tree topology it can be again seen that the performance of the cloud based controller is better than the local tree based setup for NOX and FloodLight. Based on the trend, it can be foreseen that the difference in performance would be more distinct in a larger simulation setup with more hosts, switches and higher volume of traffic where the performance of the local controller will continue to deteriorate after a certain threshold load whereas the cloud based controller will continue to maintain optimal level of service by transparently scaling computing resources in the background.

Here again, POX's performance is worst among the three controllers but the percentage change in delay is smaller. . This however does not mean that POX is more scalable in an enterprise environment because running a separate instance of the controller per virtual switch will create resource and performance bottleneck in other areas.

These set of experiments showed the advantages of running a cloud based SDN controller leveraging on the elasticity of the compute power of the cloud. It also demonstrated the general trend of the comparative performance between the controllers in terms of delay, the impact of the topology on the performance and in the case of POX the additional advantage of running a separate instance of the controller per open vSwitch.

In the above experiments, the objective has been to demonstrate the advantages of a cloud based SDN topology with the controller hosted on the cloud compared to a locally hosted controller. As an extension of this work, we further investigate the performance of these three categories of controllers solitarily decoupled from other non-controller related delay factors such as packet-in processing delay at the switch, topology related additional load and delay, etc. In the case of this experiment set B, Cbench was connected directly to the controller eliminating any external interference.

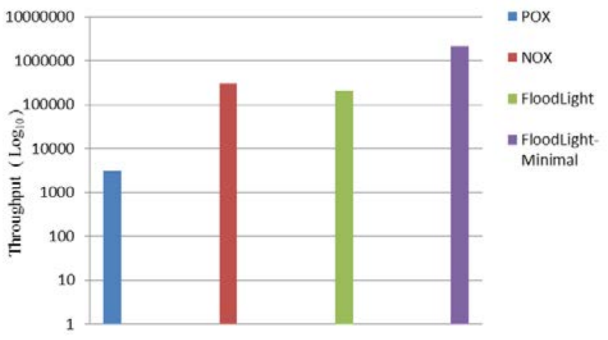

Figure 5. Throughput of the controllers.

Fig. 5 shows the mean throughput of the three controllers in terms of rate at which it can process the packet-in messages and generate the flow-mod replies. The y axis represents the throughput in $\log_{10}$ scale. It can be seen that POX's performance is minimal compared to the other controllers which makes POX implementation unsuitable for any enterprise scale large deployment. With only an average of 3087 flows/second, this low throughput is totally correlated with the high latency experienced in the previous test (Set A). It was however surprising to see that NOX's performance superseded FloodLight by a large margin. On further investigation of the architectures of NOX and FloodLight it was found that NOX's agility and performance superiority is because of its C++ core and sleek architecture with limited add-on modules. On the other hand, FloodLight has advanced mechanisms to handle a large scale network functionalities such as a throttling mechanism, an internal representation of a lambda OpenFlow switch to foresee its capacity, firewall and other advanced security modules, etc. These features impacted FloodLight performance.

For the purpose of demonstration only, all the non-mandatory modules of Floodlight were deactivated and the experiment was rerun. As seen in the 4th bar in Fig. 5, the performance is significantly higher now compared to its prior performance as well as to NOX. This shows the impact of the additional modules on the performance. However, as seen in the results for experiment Set (A) the FloodLight controller is robust in handling complex network topologies and on leveraging on the cloud based resources to provide an enterprise scale reliable and scalable performance.

The above comparison results and the analysis of the architecture is summarized in Table1.

	POX	**NOX**	**FloodLight**
Language	Python	C++	Java
Speed	Slow	Fast	Fast
Complexity	Simple	Moderate	High
Usage	Rapid prototyping	Small enterprise, R&D	Medium to large enterprise
Architecture	Simple forwarding functionalities	Can be integrated with C++/Python modules	Complex with several libraries for enterprise scale performance

Table 1. Comparison of the three controllers.

VI. CONCLUSION

The paper investigated the suitability of running a remote cloud based SDN controller. The results were validated by comparing three different topologies and with three different controllers. Although the specific performance is dependent on several other factors too outside the scope of this work, but the high level trend have been evaluated. In all the experiments it was found that the cloud based controller showed a general trend of consistent performance compared to the other topologies. It is evident from the results that with a more complex enterprise scale load and topology the cloud based SDN network model will scale up and provide a reliable performance compared to any locally hosted controller. Based on this trend, in the future further control functionalities from the switches can also be virtualized and ported to the cloud seamlessly. Some discussions on rebalancing the load between the two planes are presented in [18, 19].

The work also highlights some of the key features and analysis of the performance between the different SDN controllers. In all the experiments, it was found that the delay for FloodLight was the minimal of the three controllers. However, in the throughput test with Cbench NOX presented better throughput than the default version of FloodLight. This was because the default version was much better optimised for production network. Its global performance in an enterprise scenario is more robust and reliable. Regarding POX's results, the controller seemed generally unsuitable for production environment, but best suited for experimentation.

References

[1] F. Hu, Q. Hao and K. Bao, “A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation,” IEEE Communications Surveys & Tutorials, vol. 16, no. 4, pp. 2181-2206, 2014.

[2] Welcome to RYU the Network Operating System(NOS). https://ryu.readthedocs.io/en/latest/, (2015) (accessed 04.11.2017).

[3] S. Scott-Hayward, "Design and deployment of secure, robust, and resilient SDN controllers," in Proc. IEEE Conference on Network Softwarization (NetSoft), London, 2015, pp. 1-5.

[4] P. Mell and T. Grance, "The NIST definition of cloud computing", 2011.

[5] K. Pande Joshi and C. Pearce (2015) "Automating Cloud Service Level Agreements using Semantic Technologies", Proc. of the 2015 IEEE International Conference on Cloud Engineering (IC2E), 9-13 March 2015, Tempe, AZ, USA, pp.416-421

[6] A.T. Velte, T.J. Velte and R.C. Elsenpeter (2009), Cloud computing: A practical approach, New York: McGraw-Hill Professional Publishing.

[7] [16] M. Eisa, M. Younas, K. Basu, H. Zhu, (2016). “Trends and Directions in Cloud Service Selection”, IEEE Symposium on Service-Oriented System Engineering, Oxford, UK, 29 March-2 April 2016, pp. 1-25.

[8] Open Networking Foundation: OpenFlow switch specification. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf, (2013) (accessed: 07.02.2017).

[9] J. Zhao, Q. Yao, D. Ren, W. Li and W. Zhao (2015) “A multi-domain control scheme for diffserv QoS and energy saving consideration in software-defined flexible optical networks,” Optics Communications journal, 341, pp. 178-187.

[10] R. Casellas, R. Martínez, R. Muñoz, R. Vilalta, L. Liu, T. Tsuritani, and I. Morita (2013) “Control and management of flexi-grid optical networks with an integrated stateful path computation element and OpenFlow controller [Invited],” Journal of Optical Communications and Networking, 5(10), pp. A57-A65.

[11] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, “Advanced study of SDN/OpenFlow controllers,” in Proc. 9th Central & Eastern European Software Engineering Conference (CEE-SECR '13), Moscow, 2013, pp. 1-6.

[12] N. Gude , T. Koponen , J. Pettit , B. Pfaff , M. Casado , N. McKeown , S. Shenker, “NOX: towards an operating system for networks,” ACM SIGCOMM Computer Communication Review, vol.38 no.3, July 2008.

[13] R. Wallner and R. Cannistra, "An SDN approach: quality of service using big switch’s floodlight open-source controller", Manoa, Hawaii, in Proc. Asia-Pacific Advanced Network 3, 2013, pp. 14-19.

[14] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross et al. "The Design and Implementation of Open vSwitch." in Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI '15), Santa Clara, CA, 2015, pp. 117-130.

[15] R. L. S. De Oliveira , A. A. Shinoda, C. M. Schweitzer, and L. R. Prete, “Using mininet for emulation and prototyping software-defined networks,” in Proc. IEEE Colombian Conference on Communications and Computing (COLCOM), Bogota, 2014, June, pp. 1-6.

[16] M. Jarschel, F. Lehrieder, Z. Magyari and R. Pries, “ A flexible OpenFlow-controller benchmark,” in Proc. European Workshop on Software Defined Networking (EWSDN), Darmstadt, 2012, pp. 48-53.

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

[18] N. Feamster, J. Rexford, and E. Zegura, “ The road to SDN: an intellectual history of programmable networks”, SIGCOMM Comput. Commun. Rev., vol. 44, no. 2, pp. 87-98, Apr. 2014.D. Y. Huang, K. Yocum, and A. C. Snoeren, “High-fidelity switch models for software-defined network emulation,” in Proc. ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN '13), New York, 2013, pp. 43-48.

[19] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey and G. Wang, "Meridian: an SDN platform for cloud network services," in IEEE Communications Magazine, vol. 51, no. 2, pp. 120-127, Feb 2013.