Designing knowledge plane to optimize leaf and spine data center

Mujahid Sultan[†], Dodi Imbuido[‡], Kam Patel[‡], James MacDonald[‡] and Kumar Ratnam[‡]

[†]Department of Computer Science, Ryerson University, Toronto, Canada

[‡]Aytra Inc. 18 Wynford Dr. Toronto, Canada

mujahid.sultan@ryerson.ca; {dodi.imbuido, james.macdonald, kam.patel, kumar}@aytra.com *

Abstract—In the last few decades, data center architecture evolved from the traditional client-server to access-aggregation-core architectures. Recently there is a new shift in the data center architecture due to the increasing need for low latency and high throughput between server-to-server communications, load balancing and, loop-free environment. This new architecture, known as leaf and spine architecture, provides low latency and minimum packet loss by enabling the addition and deletion of network nodes on demand. Network nodes can be added or deleted from the network based on network statistics like link speed, packet loss, latency, and throughput.

With the maturity of Open Virtual Switch (OvS) and Open-Flow based Software Defined Network (SDN) controllers, network automation through programmatic extensions has become possible based on network statistics. Separation of *control plane* and *data plane* has enabled automated management of network and Machine Learning (ML) can be applied to learn and optimize the network.

In this publication, we propose the design of an ML-based approach to gather network statistics and build a *knowledge plane*. We demonstrate that this knowledge plane enables data center optimization using southbound APIs and SDN controllers. We describe the design components of this approach - using a network simulator and show that it can maintain the historical patterns of network statistics to predict future growth or decline. We also provide an open-source software that can be utilized in a leaf and spine data center to provide elastic capacity based on load forecasts.

I. INTRODUCTION

A. Background

The traditional design of IP networks is decentralized - for resiliency. Networking devices (e.g., switches and routers) are designed to be independent - by coupling the routing and the data. Performance is mainly achieved by merchant silicon or custom application-specific integrated circuits, thus leading to inflexible control of the devices. Recently with the ONF [1] consortium's push to produce networking devices equipped with south-bound APIs (OpenFlow [2]) the networking devices have become more accessible. Traditional switches used to have a single unit responsible for forwarding policy and a physical layer which carried out these policies. With the advent of OvS [3], OpenFlow and SDN controllers [4], the physical layer called *data plane* is separated from the policy layer called *control plane* - opening new opportunities for automation.

Maturity of Open Virtual Switch (OvS) has led to the development of many enterprise-class SDN controllers also known as network operating systems (NOS), for example, NOX [5], ONOS [6], OpenDaylight [7] and Ryu [8]. These SDN controllers separate policies from the data.

The *data plane*, responsible for forwarding packets, relies on the *control plane* for the forwarding rules or policies. The *control plane* keeps and manages policies, which are being called *flow rules*, traditionally known as forwarding rules. These *flow rules* dictate the networking policies to the *data plane*. The *data plane* is kept unaware of the network topology and relies on the *control plane* to populate their forwarding tables. Though this is a fairly new paradigm, however, it has seen success in many business domains [9], in the form of policies managed and deployed to hundreds of network devices from a single centralized *control plane*.

When the policies are developed independently of the historical network traffic patterns, these are called *static policies*. If the policies are learned based on the historical network traffic, these are called *dynamic policies*. In this publication, we stream network performance metrics to Machine Learning (ML) layer and build *dynamic policies* based on predictions by the neural network. We use these *dynamic policies* to add and or remove spine nodes to achieve desired network topologies in the leaf and spine data center.

B. Motivation

Dynamic policies have been applied to the network devices using numerous network optimization and improvement tasks such as network availability [10], MPLS optimization [11], load balancing in data centers [12], and resilience for smart grid communications [13]. To date, there has not been any significant development that uses ML in developing the *dynamic policies*.

In this study, we created a spine and leaf network and learned network traffic patterns using ML. Based on patterns found in the ML layer, we update flow rules in real-time using southbound APIs to optimize the network.

Road map: The rest of the paper is organized as follows: Related work is summarized in Section II. In Section III we describe experimental setup for network simulation, control plane and the forecast engine. In Section IV we discuss the results and in Section V we present some future directions.

II. LITERATURE REVIEW

A comprehensive list of SDN controllers and their performance and maturity comparison is given by [14], [15], and

an overview of knowledge defined networking can be found at [16]. Monitoring latency with OpenFlow is investigated by [17]. Data path performance of the spine-leaf data center is described by [18]. An OpenDaylight based MPLs controller is investigated by [11]. To improve the quality of service, [19] used leaf and spine architecture. Long short-term memory (LSTM) [20] based traffic prediction method to assist light path reconfiguration in hybrid data center networks is investigated by [21]. None of these studies investigated network optimization using LSTM from the entire metric set of the leaf and spine data center perspective nor implemented any streaming engine.

III. EXPERIMENTAL SETUP

Monitoring latency with OpenFlow was investigated by [17]. We use this method and software in our experiments. To conduct our experiments, we simulated a leaf and spine network using Mininet [22], as shown in Fig. 1. We streamed the spine switch layer traffic to a Kafka [23] server. In the ML layer, we used the LSTM neural network-based application, which consumes Kafka streams to predict spine node latency. We are developing knowledge plane components to deploy *dynamic policies* to the network via southbound APIs. The design components of our application and the data center simulation are briefly described below.
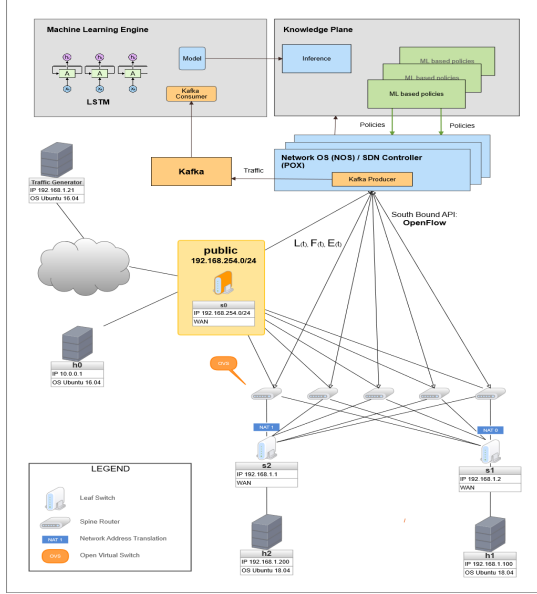

Fig. 1. The Architecture of a leaf and spine network with two "leaf" and five "spine" nodes. We use the POX SDN controller with southbound OpenFlow based APIs to access the network. Network traffic is streamed to the ML layer using Kafka topics. LSTM neural network is used to interpret and forecast network metrics. The inference is passed to the Knowledge Plane which controls the network in real-time (by adding or deleting nodes).

A. The network

Mininet[1] is a network emulator to mimic large networks on a single computer or virtual machine developed by [22]. We created a leaf and spine network segment with five spine and three leaf nodes using Mininet and transferred some large files between leaf nodes to generate the traffic.

[1]Available at https://github.com/mininet

B. The SDN Controller (POX)

We used POX[2], a python implementation of NOX [5] SDN controller which is based on the OvS [24]. A python script integrates POX to capture and stream network traffic. We used the Kafka-python[3] to generate traffic topics, which are consumed by the ML layer.

C. The Forecast Engine

The *Kafka-python consumer* is used to gather network traffic streams. We used the LSTM neural network for predicting the network metrics. Fig. 2 shows a single layer of the typical LSTM. We experimented with several architectures of LSTM to lower the validation error for the network latency. Below we briefly describe the input of the LSTM for our network simulation.

The Latency of a *link* at time t is defined by vector L_t, the *fabric link speed* by vector F_t and edge or *leaf link speed* by vector E_t giving us time series for each link in the spine network. These metrics can be written in vector form as:
$L_t = \left[L_{t-n}, L_{t-(n-1)}, \ldots, L_{t-2}, L_{t-1} \right]$,
$F_t = \left[F_{t-n}, F_{t-(n-1)}, \ldots, F_{t-2}, F_{t-1} \right]$ and
$E_t = \left[E_{t-n}, E_{t-(n-1)}, \ldots, E_{t-2}, E_{t-1} \right]$. And in matrix form for each spine *link* can be written as:

$$\begin{bmatrix} L_t \\ F_t \\ E_t \end{bmatrix} = \begin{bmatrix} L_{t-n}, L_{t-(n-1)}, \ldots, L_{t-2}, L_{t-1} \\ F_{t-n}, F_{t-(n-1)}, \ldots, F_{t-2}, F_{t-1} \\ E_{t-n}, E_{t-(n-1)}, \ldots, E_{t-2}, E_{t-1} \end{bmatrix}$$

A three-dimensional tensor represents the number of links with the third dimension for the switch label. Each input is normalized separately, and a different loss function is used, as the link latency is measured in different units than the speed. We implemented a 1-D convolutional net layer to sample the network traffic. Different architectures were tried. We used two-layered stacked LSTM with a random-dropout layer designed as three inputs and one output, as shown in Fig. 2. To train the model we used Keras [25] v2.3.0 python library with TensorFlow [26] v2.1 backend.
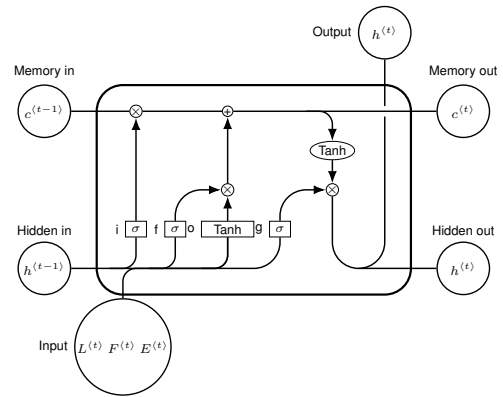

Fig. 2. Design of a single LSTM unit with the input of three network metrics (L_t, E_t, and F_t) streamed from the python-Kafka engine.

[2]available at: https://github.com/noxrepo/pox
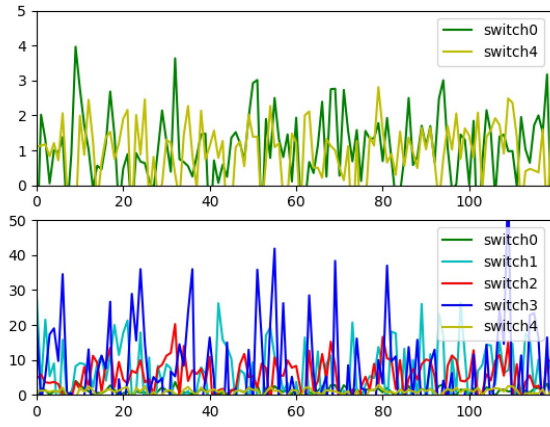[3]https://kafka-python.readthedocs.io/en/master/

Fig. 3.   Top - Switches with low predicted latency; Bottom - Prediction of hourly latency by LSTM for all switches for 120 hours (microseconds). Switches with low predicted latency (switch 0 and 4) can be removed from the network without any impact on the overall health of the network.

IV. RESULTS AND USAGE

Fig. 3 (bottom part) shows the predicted average latency of the entire network for 120 hours (5 days). In the top part of Fig. 3, two low latency switches are shown, which are good candidates to be programmatically removed by the SDN controller - to save cost without any significant impact on the network health (switches below 6 microseconds latency). A similar analysis can be done if the overall predicted network latency goes beyond some predefined thresh-hold to add new nodes in the network. This programmatic addition and deletion of the network resources can make the leaf and spine networks cost-effective.

In recent years, companies are opting for a more disaggregated approach to network equipment and software. The decoupling of the hardware and software has enabled companies to implement the best of the breed hardware with the most suitable software stack to avoid the common problem of vendor lock-ins. This decoupling has become the basis of improving architecture agility. The architecture and design of the ML-based control plane, described in this publication, enables companies to achieve the desired architectures.

V. CODE AVAILABILITY AND FUTURE RESEARCH

Python code for data center simulation on Mininet using OvS, POX SDN controller, Kafka streaming engine, and LSTM layer can be found at https://github.com/SultanMu/leaf-spine-knowledge-plane.git. We are currently developing southbound SDN components to enforce dynamic policies to the network. This work has also opened other opportunities, with the availability of vXlan [27], we plan to learn and predict *the edge* [28] workloads and add/remove nodes (eastbound or westbound) in real-time.

REFERENCES

[1] O. S. Specification, "v1. 5, open network foundation, september 27, 2013."
[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] O. S. S. ONF, "1.3. 0," 2012.
[4] K. Kirkpatrick, "Software-defined networking," *Communications of the ACM*, vol. 56, no. 9, pp. 16–19, 2013.
[5] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.
[6] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1–6.
[7] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*.   IEEE, 2014, pp. 1–6.
[8] K. Morita, I. Yamahata, and V. Linux, "Ryu: Network operating system," in *OpenStack Design Summit & Conference*, 2012.
[9] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn," *Queue*, vol. 11, no. 12, pp. 20–40, 2013.
[10] G. Nencioni, B. E. Helvik, A. J. Gonzalez, P. E. Heegaard, and A. Kamisinski, "Impact of sdn controllers deployment on network availability," *arXiv preprint arXiv:1703.05595*, 2017.
[11] E. Husni and A. Bramantyo, "Design and implementation of mpls sdn controller application based on opendaylight," in *2018 International Symposium on Networks, Computers and Communications (ISNCC)*.   IEEE, 2018, pp. 1–5.
[12] T. Kim, S.-G. Choi, J. Myung, and C.-G. Lim, "Load balancing on distributed datastore in opendaylight sdn controller cluster," in *2017 IEEE Conference on Network Softwarization (NetSoft)*.   IEEE, 2017, pp. 1–3.
[13] A. Aydeger, K. Akkaya, and A. S. Uluagac, "Sdn-based resilience for smart grid communications," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*.   IEEE, 2015, pp. 31–33.
[14] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "Sdn controllers: A comparative study," in *2016 18th Mediterranean Electrotechnical Conference (MELECON)*.   IEEE, 2016, pp. 1–6.
[15] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.
[16] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 2–10, 2017.
[17] K. Phemius and M. Bouet, "Monitoring latency with openflow," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*.   IEEE, 2013, pp. 122–125.
[18] M. Alizadeh and T. Edsall, "On the data path performance of leaf-spine datacenter fabrics," in *2013 IEEE 21st annual symposium on high-performance interconnects*.   IEEE, 2013, pp. 71–74.
[19] K. C. Okafor, I. E. Achumba, G. A. Chukwudebe, and G. C. Ononiwu, "Leveraging fog computing for scalable iot datacenter using spine-leaf network topology," *Journal of Electrical and Computer Engineering*, vol. 2017, 2017.
[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[21] H. Shi and C. Wang, "Lstm-based traffic prediction in support of periodically light path reconfiguration in hybrid data center network," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*.   IEEE, 2018, pp. 1124–1128.
[22] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, 2012, pp. 253–264.
[23] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, vol. 11, 2011, pp. 1–7.
[24] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 117–130.
[25] F. Chollet *et al.*, "Keras," https://keras.io, 2015.
[26] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.
[27] M. Mahalingam, D. G. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks." *RFC*, vol. 7348, pp. 1–22, 2014.
[28] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.