Making Queueing Theory More Palatable to SDN/OpenFlow-based Network Practitioners

Jordan Ansell, Winston K.G. Seah, Bryan Ng and Stuart Marshall
School of Engineering and Computer Science
Victoria University of Wellington, Wellington 6140, New Zealand
Email: {jordan.ansell, winston.seah, bryan.ng, stuart.marshall}@ecs.vuw.ac.nz

***Abstract*—Software-Defined Networking (SDN) is an emerging networking technology that has attracted intense interest from both the industry and research communities. Thus far, it is primarily applied to datacenters and research network environments. Despite meticulous effort in planning and equipment selection prior to deployment, there remain unknowns that can affect the network's performance after equipment has been deployed and is fully operational. Network administrators and planners would benefit from a tool that is able to monitor the load on various network entities and visualize this in real-time and, even better, predict likely performance changes arising from traffic variation; this allows them to make prompt decisions to prevent seemingly small hotspots from becoming major bottlenecks. In this paper, we present a network visualization and performance prediction tool that enables network planners to examine how their networks' performance will be affected as the traffic loads and network utilization changes. This is a first of its kind where performance prediction is based on queueing analytic models of the network configuration coupled with real-time measurements taken from the network devices.**

I. Introduction

In Software-Defined Networking (SDN), the separation of the control plane from the data plane brings about new flexibility in the routing of flows through the network. Originally intended as a means to enable researchers to conduct experiments in production networks [1], this capability also enables the implementation of virtual networks, user mobility, new network and transport layer protocols, etc. The envisaged ease of using SDN to implement, deploy and test new ideas has driven the SDN community to focus on experimentation and prototyping for the validation of new network designs, protocols and mechanisms. While this experimental approach has merits, it lacks the ability to provide insights on scalability and performance on a large scale. Key decision makers responsible for deploying networks and making architecture decisions would benefit from a performance evaluation tool that brings fundamental mathematical analysis methods such as queueing theory closer to them.

Queueing theory is a well established branch of analysis frequently used and well suited to study networks. Use of such mathematical tools on existing networks could also provide insight into how the performance changes as traffic volumes change. Queueing theory studies workloads in a variety of network traffic scenarios and offers an assortment of tools allowing for characterizing network performance. Queueing theoretic modelling techniques can contribute to accurate performance evaluation of SDN-based production networks that are exposed to new user demands continually. Accurate performance evaluation facilitates early identification of potential traffic hotspots and/or bottlenecks which carriers and network providers can quickly remedy before they become problems; likewise, yet-to-be-deployed network configurations can also be evaluated with the same level of rigour.

Analysis and modelling are the means toward determining what the expected performance and limitations of real world systems are. In networking, the limitations of scale and bottlenecks of a particular network can be determined with analytical models (without simulation or physical deployment). However, there is a steep learning curve that limits the use of queueing analysis to a handful of network practitioners. In this paper, we present a *network visualization and performance prediction* (NVPP) tool that aims to bring queueing theory closer to software-defined network practitioners. Our ability to bridge this gap is facilitated by the programmability and logically centralized control architecture of SDN [2]. To the best of our knowledge, this is the first such attempt to incorporate queueing models into a tool for predicting network performance.

II. Queueing Theory, Network Monitoring & Performance Prediction

Queueing theory has a well established record for successfully modelling networks and computing systems [3]. However, in the case of SDN, despite the extensive global research and development efforts by both academia and industry, there has been very little work done on analytical performance evaluation of SDN and/or OpenFlow [4] based networks, as noted by recent surveys, e.g. [5]. This can be partly attributed to the envisaged ease of implementing, deploying and testing new schemes that has driven the SDN community to focus on experimentation and prototyping of new protocols and mechanisms. Several notable published research articles using queueing theory to analyze SDN/OpenFlow-based networks and systems appear in [6]–[9]. However, these papers deal with the mathematical derivation and proofs of validity and make little effort to appeal to network practitioners. More effort is needed to make these results useful to network practitioners.

978-1-5090-0223-8/16/$31.00

There are many tools to assist network administrators in monitoring and visualizing the performance of a network. These existing applications display the network topology with device and performance information. Generally agents are deployed within the network to perform the monitoring. Agents are applications which collect device information, then communicate with a central server that brings all the data together and displays them. Communication between agents and central server is usually done using the Simple Network Management Protocol (SNMP) [10]. We now briefly describe a selection of such tools, which is by no means comprehensive.

Ganglia [11] is designed for monitoring clusters and grids, and the load is averaged across the whole grid. Application and device performance is monitored using software agents installed within the network. This tool is free and open source. *Oberservium* [12] is a network monitoring tool with many features. It displays network load graphs over time for each network device and collects statistics. It is free but priority access to updates comes with a fee. *Nagios* [13] is a popular network and device monitoring application that offers a live view of the network performance and enables a user to plan infrastructure upgrades. The core is open source, but for advanced features several licensed applications are available for a fee. Solarwinds [14] is a network performance monitoring application that takes live readings from the network. It can show statistical performance baselines based on historical data. Lastly, HP's Network Node Manager i (NNMi)] [15] is a large commercial tool, touted as a "comprehensive network management solution". It is used for troubleshooting and monitoring network devices, and offers performance prediction based on historical data. The key features of these tools are summarized in Table I.

Comparison of network management software

Application	Uses SNMP	Own Agent	Performance Prediction	Price
Ganglia	No	Yes	No	Free
Oberservium	Yes	No	No	Free / Paid
Nagios	Yes	Yes	No	Free / Paid
Solarwinds	Yes	Yes	No	Paid
HP NNMI	Yes	Yes	Historical	Paid, Proprietary

TABLE I

Predicting network performance has been attempted in the past where the approaches centred on analyzing historical data and making statistical inferences [16] or employing artificial intelligence techniques [17]. As networks grow in size, it has become even more difficult to collect data and monitor them, let alone predict how they will perform as traffic increases or topology changes. None of these prediction efforts nor the available network management tools, thus far, have predictive capabilities to show how a change in traffic will affect performance of network devices, or can anticipate the effect of adding new devices to the network. Existing tools focus instead on end-to-end performance and network-wide bandwidth. As SDN gains traction, network practitioners want to better understand how their yet-to-be-deployed networks or new network configurations will perform. While emulation tools like Mininet-HiFi [18] are commonly used for this purpose, there have been reported cases of poor network behaviour that manifests only in emulation but also a real deployment suffering from issues not apparent in emulation [19].

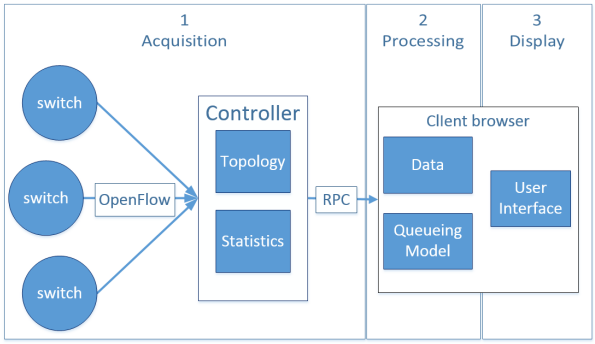

Fig. 1. NVPP Architecture

III. NVPP Overview

Common to any network monitoring application, there are generally three stages: data acquisition, processing and display, as shown in Figure 1. The SDN controller can be seen as a network monitoring tool that aggregates network information, which can be used as inputs into queueing models to predict network performance. For example, the Ryu OpenFlow Controller [20] comes equipped with tools for collecting switch and link statistics, and a network topology viewer. Our goal then is to enhance this capability (of giving the user a live view of the network's performance) with the ability to see the effect on the network's performance as a result of: (i) artificially altering network traffic; (ii) adding new switches/devices to the network; and (iii) experimenting with different queueing models.

We build our NVPP tool as an application on top of the Ryu controller (one of few supporting OpenFlow 1.3 [21] and has been tested with a wide range of OpenFlow switches.) Data acquisition is done by the controller, while the NVPP handles the processing and display, in a client-server approach. A unique feature of NVPP is ability to accept user inputs (e.g. new traffic loads, modifications to device/network configurations, etc.) and apply queueing models to calculate performance measures.

A. Design Considerations

An OpenFlow controller is generally run on commodity hardware servers (within a desktop operating system like Linux or Windows) rather than vendor specific platforms. These servers can be located separate from network administrators, and the NVPP application needs to work alongside the controller in this environment and efficiently extract data from the controller, even when accessed remotely. The controller is a central point in the network, making the forwarding decisions for the switches in the network. Where the processing stage is performed is a key performance concern, as it must not become a performance bottleneck by running applications on

it unnecessarily. Hence, together with the visual and interaction processing, the data processing and modelling are done remotely, following a client-server model. For the client, the choice of a browser-based approach is motivated by ease of implementation, portability and platform-independence. Data (network statistics) are retrieved from the controller using remote procedure calls (RPC) to avoid the need for the client to repeatedly poll the server.

B. Queueing Models' Inputs

Here, we outline the integration of queueing models into the NVPP application. The simplest queueing models (e.g. $M/M/1$ queue [3]) require an arrival rate (λ) and a service rate (μ) to derive performance results. In the case of a network, more information can be utilised, such as where packets go when they leave a node. Below is a list of information useful for an SDN queueing model:

(1) Arrival rates of traffic to each device
(2) Service rates of each device
(3) Probability of traffic passing from node i to node j
(4) Probability of no flow table entry in a switch, P_{nf}
(5) Maximum buffer sizes in each device

Dynamically changing values, viz., items (1), (3) and (4) above are available through the OpenFlow Protocol. OpenFlow does not provide access to the service rate of a device, nor does it allow the buffer size of a network device (i.e. switch) to be accessed. The service rate is not easily inferred from data sheets nor device specifications; using a previously adopted procedure [6], we measured the service rates for both physical and virtual switches available to us. The size of the device buffers could be set by a network administrator within the device, so these could be entered/added manually.

C. Port vs. Flow Statistics

Port statistics give the total flow volume in and out of a port on a device. Port statistics give both an aggregate view of the traffic flow of a device and the probabilities of a port being used. OpenFlow 1.3 also provides statistics counters on the individual flow rules within, allowing statistics of flows to be requested. The use of flows as a basis for the arrival rate data also poses three challenges: (i) how to interpret the destination of traffic from each flow's rule without access to the switch logic, as not all rules provide this information; (ii) the overhead of monitoring a large number of destinations instead of just a finite number of ports; and (iii) flows are ephemeral, and thus the information on each is not always available – some may simply vanish. While access to flow counters provides additional information and a monitoring tool could use flow statistics to estimate an average end-to-end performance, using port statistics can reliably determine the input rate of the network devices, even as the installed flows change.

IV. NVPP Design

The design of the NVPP application is divided into the server and client parts, as shown in Figure 2. Here, we outline the key components or 'objects' in object-oriented methodology.

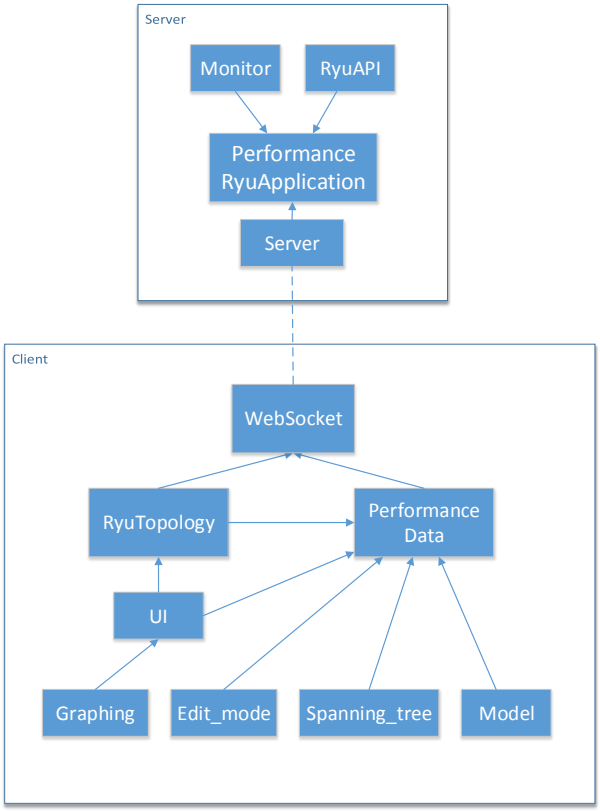

Fig. 2. Overview of Application

A. Server Part:

The object 'Performance RyuApplication' accesses the SDN network's switch statistics using the Ryu API and OpenFlow Protocol. It keeps a record of all the switches connected to the controller. Within the application is a 'monitor' object which sends out a request at a predefined interval (user configurable, currently set to 1s) for the OpenFlow statistics information from the switches. When these 'monitor' objects reply, the performance application receives the data, creates an object for each switch, and stores these in a dictionary object referenced by switch identifier (ID). Port statistics are requested from each switch in the network and the returned statistics are:

1) Number of packets received since the port became active;
2) Number of packets sent since the port became active; and
3) Duration of time the port has been active.

Additionally, the number of *packet_in* requests received by the controller are recorded. These requests are sent by switches to the controller with headers or full packets from the network data plane to process (note: *packet_in* requests do not include the statistics requests, as these operate on a different channel.) Requests are recorded per switch and also sent to the client. This allows the probability of the controller being consulted by a switch (P_{nf}) to be accessed on the client, and indicates how often the controller is being used; this is useful for more advanced queueing models of an OpenFlow network [22].

B. Client Part:

The client objects are embedded within a web browser which connects to the server, via the 'WebSocket' object. The client objects are all defined in the JavaScript downloaded from the server. Using the client web browser for displaying the visualization and interaction simplifies the setup from the user's perspective, given that web browsers are ubiquitous nowadays.

When updates arrive (from the network via the controller), an RPC calls the appropriate function for handling the data sent, whether it be performance or topology data. The performance statistics data RPC calls a function within the 'Performance Data' object with the new statistics from the switch. Switches whose data is not already stored have new performance node objects created – one node for the switch's local data and one for the live data about the switch from the network. A switch's (local) data node carries user manipulated information, such as, the queueing model to apply to that node, the brand of the switch, whether the switch is an actual network switch (or arbitrarily added "imaginary" switch) and any adjustments made on it. The live data node is where updates from the network are stored once processed.

The updates carry information about the traffic passing through a switch in the network. These nodes are stored in a dictionary referenced by a switch's ID. Performance results are computed based on a combination of live network readings and the user adjusted values. Controller information, including that required to calculate the switch P_{nf} values and the topology information also arrive via 'WebSocket'. Controller information is simply combined with the existing live information for each switch. The topology information is processed and transformed into an object which represents the network topology ready for display. Following an update of the performance information, the user interface ('UI') object is called upon to update the display. The 'UI' object reads the service rate and aggregate arrival rate values from the 'Performance Data' object, then uses this information to calculate the current performance data using the 'Model' interface. Graphs are then constructed using these two sets of data and displayed alongside the topology image, for example, as shown in Figure 3 where each column in a graph represents one switch.

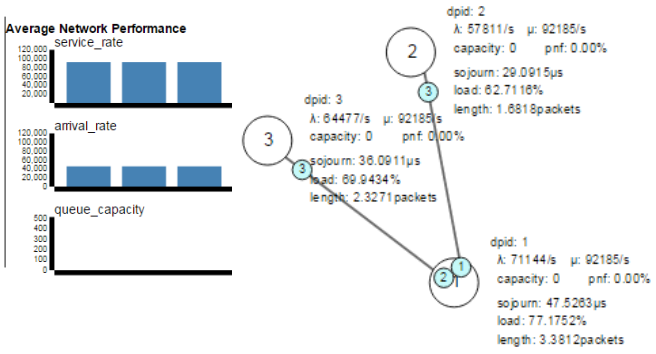

Fig. 3. Network Topology and Performance Graphs

C. Changing Device Parameters

Altering the traffic in a single node and deriving the performance measures of that node for the new values is straightforward. Our goal for NVPP goes beyond a single node, to anticipate the effect of a change on the rest of the network. Through the separation of live data and artificial adjustments, NVPP propagates the change throughout the network in a breadth-first manner. To traverse the network, a spanning tree rooted at the user-adjusted node is created. The creation of this tree requires information of: (i) node's neighbours; (ii) each neighbour's associated port; (iii) proportion of traffic that exits the node through a neighbouring node's port; and (iv) proportion of traffic that exits via other means (i.e. traffic that is destined for a connected host.) After the creation of a spanning tree, changes entered by the user are added, beginning at the root node. The effect must then be divided up among the ports.

To accurately represent the traffic flows existing in the network, the proportion of traffic currently leaving each node is used to determine how much of the adjusted traffic to pass on to the neighbouring nodes. For finer control, the distribution of traffic exiting a node could also be altered on a node-by-node basis, or the proportions of traffic travelling out each port. Alternatively, for simplicity and testing purposes, "new" traffic leaving a node can also be evenly distributed among the ports.

D. Changing Network Topology

Enabling users to alter the topology of a network can allow them to foresee how adding a new device with its own extra traffic could influence the rest of the network. A user is able to add new switches and define the traffic that it directs into the network. A user is also able to alter the proportions of traffic through the neighbouring switches to send traffic through an artificial modelled switch. In order to add devices to the topology, the user enters the 'edit' mode. In edit mode, updates from the server are paused and the present state is saved, allowing it to be restored upon exiting the edit mode.

When adding a device, it needs to be added to both the data model and the topology model, i.e. both Performance Data and Topology objects. A device will not initially have connections to other switches, these must be added manually. Each time a link is added, the ports within the data model are updated and links in the topology model are created; the opposite is true for removing a link. The user will need to alter the proportion of traffic flowing into the additional switch from the adjacent switches, these adjustments are performed directly on the live data within the data model and not through 'adjustments'.

Traffic flowing through the added node is defined purely through the adjustment mechanism, and this behaves as it would in a regular adjustment, any artificial nodes are transparent to the Spanning tree object. When the user has completed the network analysis and performance prediction exercise, he exits the edit mode which restores the original objects to their previous state, then awaits a fresh performance update from the server to resume the live view of the network performance.

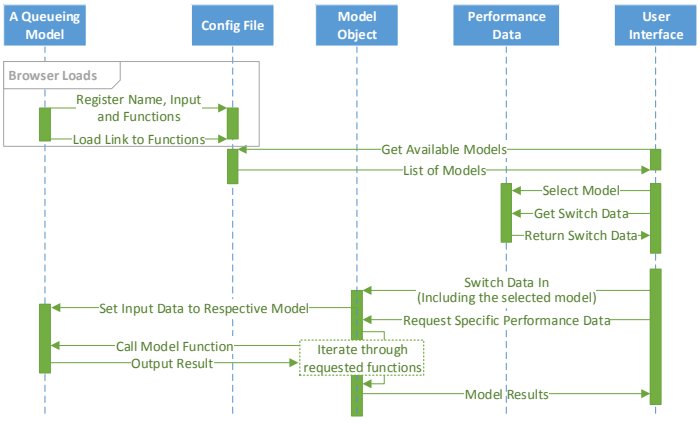

Fig. 4. Sequence Diagram Showing Queueing Model Interface (Object)

E. Interface for Queueing Models

Key features of NVPP are: (i) the choice of different queueing models to apply on network devices; (ii) the ability to modify various parameters; and (iii) the modular interface for queueing models. The modular interface permits addition of new queueing models of switches and devices, i.e. mathematical equations representing these queueing models. This poses an interesting design challenge. An interface which supplies the selected queueing model (representing a network device) with all the data available, and simply handles any data it returns is used. This produces charts and graphs to be created for comparing the performance of different network devices. The relationship between the application and queueing models is shown in Figure 4. An intermediary object is used to stage the model code and provide a place for the model to declare its functionality to the rest of the application. The user interface uses this declaration to supply the user with a choice of models on each switch. When the performance is calculated, the model interface uses the selected model on each switch to determine the code to be used. New queueing models can be added as objects (e.g. "A Queueing Model") which will be automatically loaded when the user starts the application.

V. System Validation and Performance

We identified three metrics to assess NVPP's performance, viz., (i) network status update delay (time needed to retrieve live network data from the server, process and display the results); (ii) creation of a spanning tree for traffic adjustments by the user (cf: Section IV-C); and (iii) applying adjustments on a switch in the network (propagating the necessary changes to the network).

A test system was implemented on a platform using an Intel i5-3317U, dual core 1.7 GHz processor with 8GB of memory, running the Windows 7 operating system, which we regard as a typical platform. In the following subsections, we discuss the experimental results for these metrics, with 95% confidence intervals. A solid red line is shown on some graphs to indicate one second, for reference.

A. Network Status Update Delay

The server part of NVPP retrieves status data of all switches from the Ryu controller (cf: Section IV-A) and updates the stored status information of all the switches administered by the Ryu controller; currently, it has been configured to do this at one second intervals. Note that NVPP pulls data out of the Ryu (SDN) controller, not the switches themselves, and the view of the network reflects that of the controller. Figure 5 shows a significant difference between the time taken for network data ('Mean Data Update Delay') to be updated and the User Interface (UI), i.e. graphical display, to be updated. Updating the network data takes less than $30\mu s$ for all measured network sizes, while the UI update delay reached ten times this at only 500 switches. While the UI update delay has been measured, it is not shown because it almost matches the total update delay times, the difference being close to the data update time. Extensive trials conducted showed that the maximum time for 2000 nodes occasionally exceeded the 1 second limit, and the confidence interval shown. It is therefore prudent to assume that NVPP can support up to 1500 nodes. From a user's perspective though, the UI topology view of a 1500-node network becomes overly cluttered and illegible.

B. Traffic Adjustment Setup and Propagation Delays

When a user changes a network device's traffic (or other) parameters to assess the impact on the network, a delay is incurred to create the spanning tree, followed by the time needed to propagate the changes to all nodes in the tree. Although we are not dealing with realtime updates, the delay must be tolerable to the user for network sizes similar to the live update scenario discussed previously. As shown in Figure 6, the more significant delay is incurred in the spanning tree creation, which on average is eight times greater than the time to propagate the changes through a created tree. They both show a polynomial increase as the size of the network grows. Despite that, a network of 2000 nodes incurs a delay of only $40\mu s$ which has minimal impact on NVPP's performance.

C. Note on Correctness of Queueing Models

NVPP does not aim to nor have the capability to provide proof of correctness of any queueing model for the network and/or switches. Rather, NVPP aims to incorporate proven (published) queueing models of SDN/OpenFlow networks and devices, e.g. [6]–[9], and make them easily usable by network practitioners. A recommended approach would be to first run the visualization function to show the network performance. Then, select a suitable queueing model from the list provided, use the current parameter values, e.g. arrival rate λ and service rate μ, and show the network performance as computed by the queueing model. An exact match between the real performance and that provided by the queueing model is unlikely; however, a network administrator can pick a queueing model that provides a close enough match and use that to predict the change in network performance arising from traffic changes.

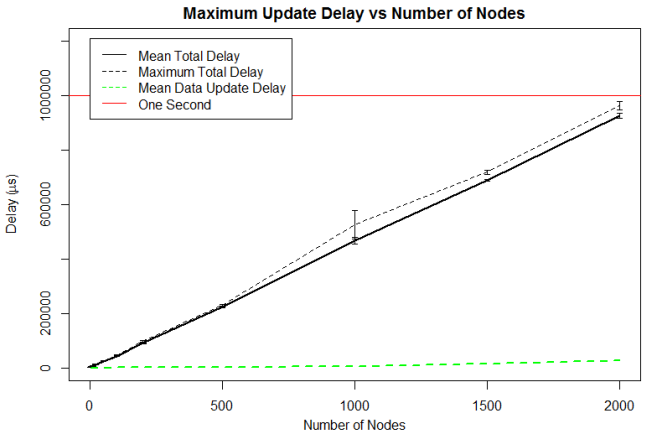

Fig. 5. Time to perform an update

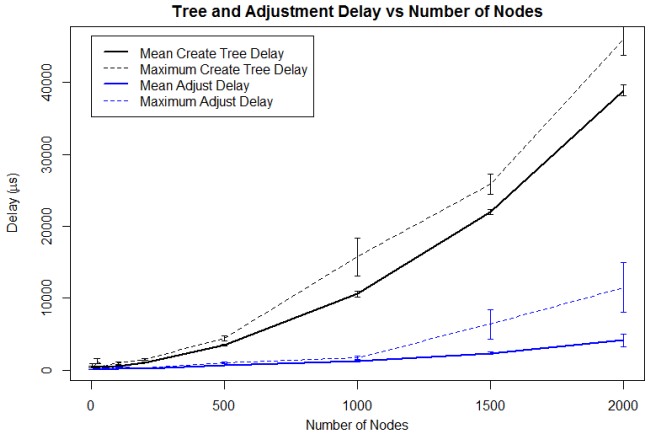

Fig. 6. Latency of creating a spanning tree and calculating an adjustment

VI. Conclusion

We have developed a *network visualization and performance prediction* (NVPP) tool that provides network practitioners the ability to predict a network's performance when traffic parameters and the network topology change, in addition to a real time view of a network along with functions provided by existing network management tools. This prediction is done through the use of analytical models derived using queueing theory. Queueing theory has been the cornerstone of network performance analysis but networks practitioners have shunned away from queueing theory because of the need to understand the complex mathematics involved. Deriving usable results from queueing models can be a laborious effort of first collecting network status data from all the network devices, pre-processing them before they can be used as inputs for a selected queueing model. The SDN paradigm gives network providers and administrators a programmatic interface to the logically centralized view and control of a network. An SDN controller periodically acquires the status of network devices within its administrative domain, which our tool exploits to gather the necessary data for input into the queueing models. Through NVPP, we hope to provide network practitioners a power tool to help them design and manage the networks effectively. A working prototype of NVPP can be obtained from http://github.com/serendipiddy/openflow-performance-visualizer.

References

[1] 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, Apr 2008.

[2] M. Jarschel, T. Zinner, T. Hoßfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, Attributes, and Use Cases: A Compass for SDN," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 210–217, Jun 2014.

[3] L. Kleinrock, *Queueing Systems – Volume 1: Theory*. Wiley-Interscience, 1975.

[4] Open Networking Foundation, "Openflow specification," https://www.opennetworking.org/sdn-resources/openflow.

[5] B. Nunes, M. Mendonca, X. Nguren, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, 2014.

[6] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and Performance Evaluation of an OpenFlow Architecture," in *Proc. of 23rd International Teletraffic Congress*, San Francisco, CA, USA, Sep 2011, pp. 1–7.

[7] J. Hu, C. Lin, X. Li, and J. Huang, "Scalability of Control Planes for Software Defined Networks: Modeling and Evaluation," in *Proc. of IEEE/ACM International Symposium on Quality of Service (IWQoS)*, Hong Kong, China., 26-27 May 2014.

[8] L. Yao, P. Hong, and W. Zhou, "Evaluating the controller capacity in software defined networking," in *Proc. of 23rd International Conference on Computer Communication and Networks (ICCCN)*, Shanghai, China, 4-7 Aug 2014.

[9] K. Mahmood, A. Chilwan, O. Østerbø, and M. Jarschel, "Modelling of OpenFlow-based software-defined networks: the multiple node case," *IET Networks*, vol. 4, no. 5, pp. 278–284, 2015.

[10] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)," *RFC1157*, May 1990.

[11] Ganglia Monitoring System. [Online]. Available: http://ganglia.sourceforge.net

[12] Observium – Network monitoring with intuition. [Online]. Available: http://www.observium.org/

[13] Nagios. [Online]. Available: http://www.nagios.org/about/overview/

[14] Solarwinds network performance monitor. [Online]. Available: http://www.solarwinds.com

[15] Hewlett Packard Enterprise. Network Node Manager i. [Online]. Available: http://www8.hp.com/nz/en/software-solutions/network-node-manager-i-network-management-software/

[16] Q. Xu, S. Mehrotra, Z. Mao, and J. Li, "PROTEUS: Network Performance Forecast for Real-time, Interactive Mobile Applications," in *Proc. of the 11th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Taipei, Taiwan, 25-28 Jun 2013, pp. 347–360.

[17] R. Wolski, "Forecasting network performance to support dynamic scheduling using the network weather service," in *Proc of the 4th IEEE International Symposium on High Performance Distributed Computing*, Portland, OR, USA, 5-8 Aug 1997, pp. 316–325.

[18] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proc of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, Nice, France, 10-13 Dec 2012, pp. 253–264.

[19] A. Roy, K. Yocum, and A. C. Snoeren, "Challenges in the Emulation of Large Scale Software Defined Networks," in *Proc. of the 4th Asia-Pacific Workshop on Systems (APSys)*, Singapore, 29-30 July 2013.

[20] Ryu SDN Framework. [Online]. Available: http://osrg.github.io/ryu/

[21] D. Kreutz, F. M. V. Ramos, P. E. V. issimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, Jan 2015.

[22] Y. Goto, H. Masuyama, B. Ng, W. K. G. Seah, and Y. Takahashi, "Queueing Analysis of Software Defined Network with Realistic OpenFlow-based Switch Model," Dec 2015, submitted to the IFIP Networking 2016 Conference.