On the Impact of Network State Collection on the Performance of SDN Applications

Mohamed Aslan, and Ashraf Matrawy
Carleton University, ON, Canada

***Abstract*—Intelligent and autonomous SDN applications need to monitor the network state in order to take appropriate actions. In this letter, we compare the impact of active and passive network state collection methods on an SDN load-balancing application running at the controller. We do this comparison through: (1) the results of a mathematical model evaluation we derive for the SDN load-balancer, (2) the results of a series of elaborate experiments we ran on our emulation setup. The results show that in case of low-variation traffic, the load-balancer with passive state collection performed better than the active one, which was confirmed by both model and experimental evaluation. However, the load-balancer with the active state collection was more resilient to the nature of the traffic load.**

***Index Terms*—SDN, SDN performance, SDN measurement, SDN applications, SDN controller.**

I. Introduction

In this letter we focus on OpenFlow-based network state collection mechanisms due to the popularity of OpenFlow in SDN-enabled devices. In particular, we compare passive network state collection maintained by the controllers with active network state collection that relies on periodically polling the switches. The ONUG [1] has recently showed interest in network state collection.

In this letter, we make the following contributions: (1) we study the impact of passive and active OpenFlow state collection mechanisms on certain key performance indicators that we define. To study this: (a) we ran simulations based a mathematical model we derived for the load-balancing (LB) application, and (b) we developed a load-balancing application and we ran experiments in the context of single and distributed controller environments. (2) We study the impact of low-variation and high-variation traffic loads on the application performance given different state collection mechanisms.

II. Background on SDN Controllers

For the rest of the letter, we refer to a physically centralized controller as a "single controller", while we call logically centralized but physically distributed controllers "distributed controllers". One major challenge is how to keep the controllers' network views consistent. With distributed controllers, as each controller collects state information about the network, they need to exchange their views in order to build a global network view. The consistency between controllers is governed by the consistency model employed [2].

Levin et al. [3] used a flow simulator to study the design of distributed SDN applications. First, they evaluated the impact of inconsistent global network view on the performance of a LB application. They showed that the inconsistency can significantly degrade the performance of SDN applications. Second, they investigated the application complexity versus robustness against outdated states. They concluded that applications that are aware of the underlying state distribution can avoid depending solely on outdated states when making decisions, and perform better than those that are unaware. Guo et al. [4] further extended the work to overcome the issues of synchronization overhead and forwarding-loops due to inconsistency.

III. Network State Collection

There exist different mechanisms that network application developers can employ in collecting the network state information required for maintaining an up-to-date network view. Excluding *middle-boxes* which are likely to face scalability issues, we classify these techniques into: (1) non-OpenFlow-based, and (2) OpenFlow-based. First, non-OpenFlow-based mechanisms include the use of other protocols as SNMP or sFlow. Second, OpenFlow-based mechanisms involve the use of information stored at the controllers as well as *flow* or *port* statistics that are polled from the switches. In this letter we are only interested in OpenFlow-based mechanisms. Further we divide OpenFlow-based mechanisms into: (1) *passive*, and (2) *active*. The choice of whether to design a network application to base its decisions on passive, active, or hybrid state collection could be a design challenge.

Passive Network State Collection. When a new packet arrives at a switch with no flow rule to match, by default, the packet's header will be forwarded to the controller. The controller is then responsible for deciding what to do with the packet and for instructing the switches. The controller can locally keep track of flow rules inserted into the switches, which we call *passive* state collection. However since by default not every packet is forwarded to the controller, the controller-maintained information are mainly flow-based information and may not include packet or byte information. Yu et al. [5] proposed a passive network monitoring approach that computes the utilization of the links between switches, and they discussed how their approach can be combined with *active* approaches.

Active Network State Collection. Controllers can asynchronously communicate with the switches they control in order to request the required state information. An example

is the flow and port statistics. OpenFlow defines the format of the messages that can be exchanged by the controllers and the switches. With regards to applications that require up-to-date state information, they need to periodically communicate with the network switches as such information could be outdated relative to the application needs. It is also important to highlight that our results (§VII-B) indicate that the period at which the controllers poll the necessary information from the switches can impact the performance of those applications.

IV. Performance Indicators

In this letter, we study the behavior of a LB application (§V) when employing different mechanisms for network state collection. For simplicity, we consider the load-balancing between two servers. We opted for the relative difference between the traffic *received* by each server as an indicator for the LB's performance. The smaller the difference, the better the performance. The relative difference can be defined in terms of flows (ξ_f) or bytes (ξ_b). We chose the relative difference, as a normalized indicator (unit-less) to be able to compare (ξ_f) and (ξ_b) where needed.

Equations 1 and 2 show the relative difference in terms of flows and bytes, respectively. In case of ξ_f, f_i represents the flow count (as measured at the server) assigned to server i (in our experiments, $i \in \{1, 2\}$) at a given time. While for ξ_b, b_i represents the amount of traffic received (in bytes, measured at the server) by server i at a given time.

$$\xi_f = \frac{|f_1 - f_2|}{f_1 + f_2} \quad \cdots \quad s.t. \quad 0 \leq \xi_f \leq 1 \tag{1}$$

$$\xi_b = \frac{|b_1 - b_2|}{b_1 + b_2} \quad \cdots \quad s.t. \quad 0 \leq \xi_b \leq 1 \tag{2}$$

V. Load-Balancer's Design

A simplified version of the load-balancing algorithm is presented in Algorithm 1. When a new client's request (packet) arrives at a switch and there are no rules in the switch's flow-table on how to process this request, the switch will forward the request to its controller. The controller will, according to its local view of the network, decide where to assign the flow associated with the request. In case of distributed controllers, a controller can assign the flows to its local domain server or forward the flows to the out-of-domain server connected to the other switch. The two controllers periodically synchronize their state. On each synchronization period, the controllers exchange their local view of network. We use a hard-time of 2 sec for all flows i.e., a flow rule lives in the switch's flow-table for only 2 sec, then it has to be reassigned.

We consider four different variation for a LB: (1) a single-controller LB that uses passive state collection (SP), (2) a single-controller LB that uses active state collection (SA), (3) a two-controller distributed LB that uses passive state collection (DP), and (4) a two-controller distributed LB that uses active state collection (DA). The objective of these LBs is to reduce the difference between the servers' link utilization. i.e. hence reduce ξ (0 is the optimum).

Algorithm 1: SDN load-balancing at the controllers.

```
Data: S_n, set of n servers.
Data: L_n, set of traffic load of the n servers (could be
      measured in flows or bytes)
begin
    while pkt arrives do
        l_min ← ∞
        foreach s ∈ S_n do
            if L(s) < l_min then
                l_min ← L(s)
                s_min ← s
        SetupPath(pkt, s_min)
```

VI. Model Evaluation

We derive ξ_f only in the case of SP. However, similar steps can be used to derive ξ_b for SA. We make the following assumptions: (1) the number of switches, servers and controllers in the network ($N = 2$), (2) the load-balancing algorithm is invoked for every flow arrival event (as in the case of OpenFlow), and (3) we ignore network delays. Next, we use the following notations:

- $E_k(t)$ — the number of expired flows assigned at server k ($1 \leq k \leq N$) at a given time t.
- L_k^i — the load on server k, i is the i^{th} flow inter-arrival event.
- Δ^i — the difference between the two servers' loads.

$$\Delta^i = L_1^i - L_2^i \tag{3}$$

- σ^i — the total loads on the two servers.

$$\sigma^i = L_1^i + L_2^i \tag{4}$$

- ξ_f^i — the relative difference between the servers.

$$\xi_f^i = |\Delta^i|/\sigma^i \tag{5}$$

- M_k^i — number of new flows that will be assigned to server k, based on the second assumption $\sum_{k=1}^{N} M_k^i = 1$.
- d^i — the decision parameter of controller.

$$\begin{aligned} d^{i+1} &= (L_1^i - E_1^{i+1}) - (L_2^i - E_2^{i+1}) \\ &= \Delta^i - E_1^{i+1} + E_2^{i+1} \qquad (using(3)) \end{aligned} \tag{6}$$

$$L_k^{i+1} = L_k^i - E_k^{i+1} + M_k^{i+1} \tag{7}$$

$$M_1^{i+1} = \begin{cases} 0, & \text{if } d^i > 0 \\ 1, & \text{if } d^i \leq 0 \end{cases}, \quad M_2^{i+1} = \begin{cases} 0, & \text{if } d^i \leq 0 \\ 1, & \text{if } d^i > 0 \end{cases} \tag{8}$$

$$\begin{aligned} \Delta^{i+1} &= L_1^{i+1} - L_2^{i+1} && (using(3)) \\ &= (L_1^i – E_1^{i+1} + M_1^{i+1}) - (L_2^i – E_2^{i+1} + M_2^{i+1}) && (using(7)) \\ &= \Delta^i + (M_1^{i+1} – M_2^{i+1}) + (E_2^{i+1} – E_1^{i+1}) \end{aligned}$$

$$\begin{aligned} \sigma^{i+1} &= L_1^{i+1} + L_2^{i+1} && (using(4)) \\ &= L_1^i – E_1^{i+1} + M_1^{i+1} + L_2^i – E_2^{i+1} + M_2^{i+1} && (using(7)) \\ &= \sigma^i + 1 - (E_1^{i+1} + E_2^{i+1}) \end{aligned} \tag{9}$$

$$\begin{aligned} \xi_f^{i+1} &= |\Delta^{i+1}|/\sigma^{i+1} && (using(5)) \\ &= \frac{|\Delta^i + (M_1^{i+1} – M_2^{i+1}) + (E_2^{i+1} – E_1^{i+1})|}{\sigma^i + 1 - (E_1^{i+1} + E_2^{i+1})} \quad \blacksquare \end{aligned} \tag{10}$$

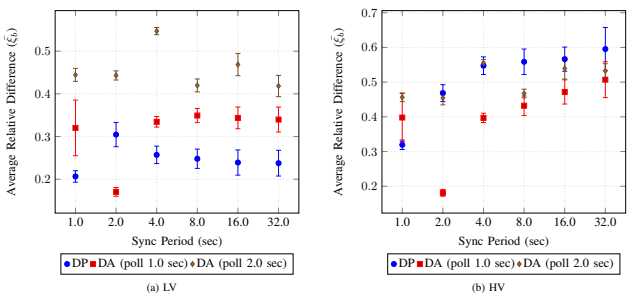

Fig. 1: Average relative difference ($\bar{\xi}_b$) vs synchronization period in case of DP and DA LBs. Results obtained using the model shown in §VI.

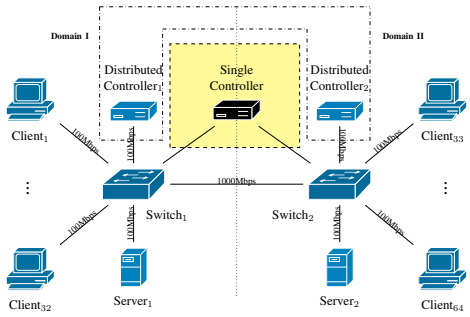

Fig. 2: The network topology used in the two scenarios: (1) a single controller was used (in black), and (2) two distributed controllers (in blue) were used.

In the cases of DP and DA, each controller maintains its own decision parameter which may differ in values. For DP controllers, they base their decision on values of L^i for the local server and L^{i-s} for the out-of-domain server, where s is a variable that depends on the synchronization period. For DA controllers, they base their decision on values of L^{i-p} for the local server and L^{i-s} for the out-of-domain server, where p is a variable that depends on the polling period.

Fig. 1 shows the effect of the synchronization period on the performance of both DP and DA. In the case of LV, DP performed better than the DA (except at sync. period 2). The effect of the polling period had a higher impact than the synchronization on DA. In the case of HV, DP was more impacted than DA with higher synchronization periods.

VII. Experimental Evaluation

A. Environment Setup

To highlight the impact of employing different network state collection mechanisms, we designed a series of experiments we ran on our emulation setup to show how a LB application running at the controller will perform when it takes actions based on both passive collection of flow information and active collection of byte information polled from-switch. We do this in the context of both a single and a distributed SDN controllers environments.

In our setup, we run POX [6] controller instances as in-band hosts emulated by Mininet. The consistency model employed in our experiment in the cases of distributed LBs (DP and DA) is known as the *delta* consistency model [7]. The choice of the right value of the synchronization period is application-specific. Using very small periods might not be feasible due to various network delays or communication overhead, while very long ones can badly hurt the performance of the application (we show in §VII-B). Also the number of synchronization messages exchanged between the controllers increases with the number controllers.

Figure 2 shows the topology used in our experiment. The topology was used in two scenarios: (1) a single controller was used (in black), and (2) two distributed controllers (in blue) were used. Two OpenFlow-enabled switches are connected via a 1000 Mb/s link. The setup also includes two servers; each is connected to a separate switch via a 100 Mb/s link. In case of the second topology, the network is divided into two domains,

TABLE I: Employed traffic loads and their parameters. r_1 and r_2 are flow arrival rates for switch 1 and 2, respectively. p_1 and p_2 are packets-per-flow arrival rates for switch 1 and 2, respectively. Payload of any packet is 4KBytes.

	LV	HV
Flows	Poisson process	Poisson process
Flows rates	$r_1 = 6f/s, r2 = 4f/s$	$r_1 = 8f/s, r_2 = 2f/s$
Pkts-per-flow	Poisson process	Poisson process
Pkts-per-flow rates	$p_1 = 34p/s, p_2 = 30p/s$	$p_1 = 48p/s, p_2 = 16p/s$

where each domain consists of: a switch, a server, a number of clients, and a controller that is responsible for controlling that domain. Finally, 64 clients (32 at each switch) are connected via 100 Mb/s links. The clients will generate UDP requests and create the traffic. We employed two traffic loads. The first we call the low-variation traffic load (LV), while the second is the high-variation traffic load (HV). Table I shows the parameters of each load.

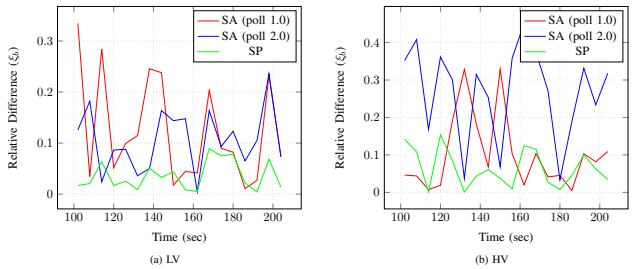

Fig. 3: Relative difference (ξ_b) vs time.

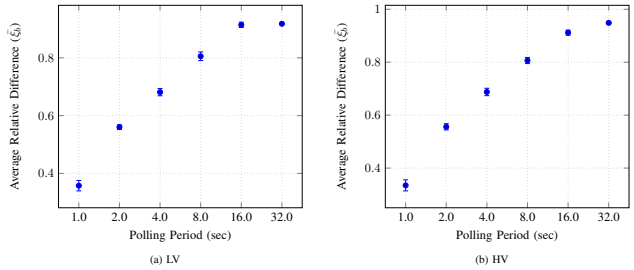

Fig. 4: Average relative difference ($\bar{\xi}_b$) vs polling period in case of a SA LB.

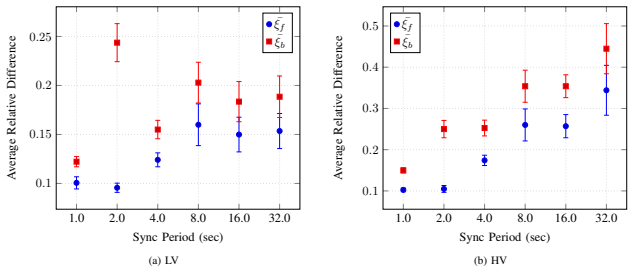

Fig. 5: Average relative difference in bytes ($\bar{\xi_b}$) and flows ($\bar{\xi_f}$) vs synchronization period in case of a DP LB.

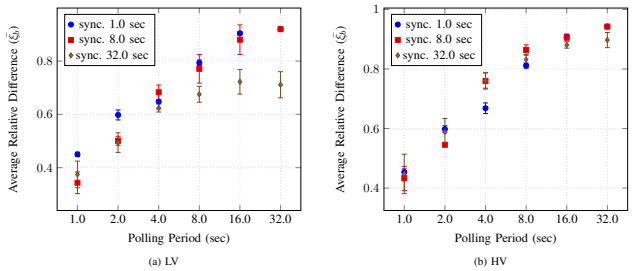

Fig. 6: Average relative difference ($\bar{\xi_b}$) vs polling period in case of a DA LB.

B. Results

We show the results of our experiments. $\bar{\xi}$ represents the relative difference averaged every 2 sec over 10 runs. Recall the definition of the relative difference ξ_f (1) and ξ_b (2). The smaller the relative difference the better the performance.

Fig. 3 shows how the relative difference in the number of bytes (ξ_b) measured at each server varies with time (6 sec averaged). SP performed better most of the time than SA (with polling period 1 and 2 sec). Comparing Fig. 3a and 3b, the results show that the traffic load had an impact that was not significant, i.e. in the LV case the SP performed better than SA even at small polling periods.

Fig. 4 shows the effect of the polling period on the performance of the SA LB. Regardless of the traffic load (Fig. 4a and 4b), SA is affected by the polling period. As the polling period increases, the performance of SA degrades.

Fig. 5 shows that the performance of DP was affected by the synchronization period among the controllers. As the synchronization period increases, the performance degrades. Levin et al. [3] showed similar results regarding flow-based controllers when measuring the performance in terms of flows. Relying on flows in decision making (i.e. passive state collection) impacts the results as follows: (1) the performance of the LB was worse in the case of the HV load than that of the LV load (higher $\bar{\xi_f}$ and $\bar{\xi_b}$), and (2) the flow-based $\bar{\xi_f}$ and byte-based $\bar{\xi_b}$ performance indicators deviated.

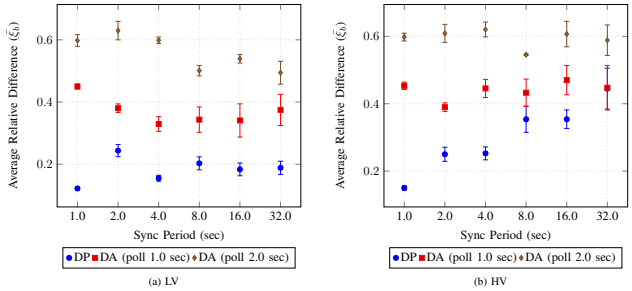

Fig. 7: Average relative difference ($\bar{\xi_b}$) vs synchronization period in case of DP and DA LBs.

Fig. 6 shows the effect of the polling period on the performance of the DA LB, at synchronization periods of 1, 8 and 32 sec. DA is affected by the polling period. Its performance degrades as the polling period increases, the same as SA (Fig. 4). Our results show that the polling period has more impact than the synchronization period on DA's performance (same as Fig. 1a). As the polling period increases, the state information used by the LB becomes increasingly outdated. Therefore, for LV traffic load, reducing the frequency of synchronization (i.e. 32 sec) limits the exchange of outdated state information between the controllers. This is demonstrated in Fig. 6a by a lower value of $\bar{\xi_b}$ in the case of a 32 sec synchronization period at high polling periods (8, 16 or 32 sec).

Fig. 7 shows the effect of the synchronization period on the performance of both the DP and the DA LBs. For LV traffic, DP outperformed DA, even at high synchronization periods (similar to 1a). However, for HV traffic the performance of DP started to degrade with the synchronization period.

VIII. Conclusion

Our evaluation shows that in case of low-variation traffic, where flows are comparable (in byte counts), the application that relied on passive state collection performed better than the one that relied on active state collection. The performance of the application that relied on active state collection was mainly dependent on the polling periods, and in the context of a distributed environment was more affected by the polling periods than the synchronization periods. Lastly, since the results show that the nature of traffic (LV versus HV) has an impact on the application performance, SDN application developers should pay attention to how they define flows in their applications.

Acknowledgment. The authors acknowledge support from the Natural Sciences and Engineering Research Council of Canada (NSERC) through the NSERC Discovery Grant program.

References

[1] ONUG Network State Collection, Correlation and Analytics Working Group, "Network state collection, correlation and analytics product / rfi requirements," 2015. [Online]. Available: https://opennetworkingusergroup.com/wp-content/uploads/2015/05/Network-State-Collection-White-Paper_2015_V5.pdf

[2] A. Panda, C. Scott *et al.*, "Cap for networks," in *Proc. of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013.

[3] D. Levin, A. Wundsam *et al.*, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proc. of the first workshop on Hot topics in software defined networks*. ACM, 2012.

[4] Z. Guo, M. Su *et al.*, "Improving the performance of load balancing in software-defined networks through load variance-based synchronization," *Computer Networks*, 2014, communications and Networking in the Cloud.

[5] C. Yu, C. Lumezanu *et al.*, "Flowsense: Monitoring network utilization with zero measurement cost," in *Passive and Active Measurement*. Springer, 2013.

[6] J. Mccauley, “Pox: A python-based openflow controller,” 2014. [Online]. Available: http://www.noxrepo.org/pox/about-pox/

[7] A. Singla, U. Ramachandran, and J. Hodgins, “Temporal notions of synchronization and consistency in beehive,” in *Proc. of the ninth annual ACM symposium on Parallel algorithms and architectures*. ACM, 1997.