Logically Centralized? State Distribution Trade-offs in Software Defined Networks

Dan Levin
TU Berlin / T-Labs
dan@net.t-labs.tu-berlin.de

Andreas Wundsam
ICSI / UC Berkeley
andi@icsi.berkeley.edu

Brandon Heller
Stanford University
brandonh@stanford.edu

Nikhil Handigol
Stanford University
nikhilh@stanford.edu

Anja Feldmann
TU Berlin / T-Labs
anja@net.t-labs.tu-berlin.de

ABSTRACT

Software Defined Networks (SDN) give network designers freedom to refactor the network control plane. One core benefit of SDN is that it enables the network control logic to be designed and operated on a global network view, as though it were a centralized application, rather than a distributed system – logically centralized. Regardless of this abstraction, control plane state and logic must inevitably be physically distributed to achieve responsiveness, reliability, and scalability goals. Consequently, we ask: "How does distributed SDN state impact the performance of a *logically centralized* control application?"

Motivated by this question, we characterize the state exchange points in a distributed SDN control plane and identify two key state distribution trade-offs. We simulate these exchange points in the context of an existing SDN load balancer application. We evaluate the impact of inconsistent global network view on load balancer performance and compare different state management approaches. Our results suggest that SDN control state inconsistency significantly degrades performance of logically centralized control applications agnostic to the underlying state distribution.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Centralized Networks; C.2.4 [**Distributed Systems**]: Network Operating Systems

Keywords

Software Defined Network, Control Plane, Sensitivity Study

1. INTRODUCTION

The emergence of Software Defined Networking (SDN) [8] has sparked significant interest in rethinking classical approaches to network architecture and design. SDN enables the network control plane logic to be decoupled from the network forwarding hardware, and moves the control logic and state to a programmable software component, the *controller*. One of the key features enabled through this decoupling is the ability to design and reason about the network control plane as a *centrally* controlled application operating on a global network view (GNV) as its input. In essence, SDN gives network designers freedom to refactor the network control plane, allowing network control logic to be designed and operated as though it were a centralized application, rather than a distributed system – logically centralized.

Thus, SDN designers now face new choices; in particular, how centralized or distributed should the network control plane be? Fully *physically* centralized control is inadequate because it limits *(i)* responsiveness, *(ii)* reliability, and *(iii)* scalability. Thus, designers resort to a physically distributed control plane, on which a logically centralized control plane operates. In doing so, they face trade-offs between different consistency models and associated liveness properties.

Strongly consistent control designs always operate on a consistent world view, and thus help to ensure coordinated, correct behavior through consensus. This process imposes overhead and delay however, and thus limits responsiveness which can lead to suboptimal decisions.

Eventually consistent designs integrate information as it becomes available, and reconcile updates as each domain learns about them. Thus, they react faster and can cope with higher update rates, but potentially present a temporarily *inconsistent* world view and thus may cause incorrect behavior. For instance, an inconsistent world view can cause routing loops or black holes.

Consequently, it is important to understand how physically distributed control plane state will impact the performance and correctness of a control application logic designed to operate as though it were centralized. Specifically, when the underlying distributed control plane state leads to inconsistency or staleness in the global network view, how much does the network performance suffer?

We approach this problem by systematically characterizing the state exchange points in a distributed SDN control plane. We then identify two key state distribution trade-offs that arise: *(i)* The trade-off between control application performance (optimality) and state distribution overhead and *(ii)* application logic complexity vs. robustness to inconsistency in the underlying distributed SDN state.

We then simulate these trade-offs in the context of an existing SDN application for flow-based load balancing, in order to evaluate the impact of an inconsistent global network view on the performance of the "logically centralized" control application. We compare two different control application approaches which operate on distributed SDN state: A simple approach that is ignorant to potential inconsistency in the global network view, and a more complex approach that considers the potential inconsistency in its network view when making a load balancing decision. In our simu-

HotSDN'12, August 13, 2012, Helsinki, Finland.

lation scenario, initial results demonstrate that global network view (GNV) inconsistency significantly degrades the performance of our network load balancing application which is naïve to the underlying distributed SDN state. The more complex application state management approach is more robust to GNV inconsistency.

We place our sensitivity study in the context of related work in Section 2. We then present the inherent state exchange points and trade-offs in SDN design in Section 3, and discuss their impact on our example application in Section 4. We quantitatively explore the trade-offs using a simulation approach in Section 5, present our experiment setup and preliminary results, then conclude in Section 6.

2. RELATED WORK

Related work falls into three main categories – (1) SDN applications that motivate our study, (2) control frameworks that provide a logically centralized view to SDN applications, and (3) previous studies on routing state distribution trade-offs.

Our work is inspired by SDN applications such as in-network load-balancing [5, 14] that require a distributed control plane implementation for scalability, but at the same time require an up-to-date view of the network to optimize their objective function. Our study explores this specific trade-off while being agnostic to control plane implementation specifics – proactive vs reactive, micro vs macro-flow management, or short vs long timescale switch-controller interactions.

Onix [7] is a control plane platform designed to enable scalable control applications. Its main contribution is to abstract away the task of network state distribution from applications and provide them with a logical view of the network state. Onix provides a general API for control applications, while allowing them to make their own trade-offs among consistency, durability, and scalability. The paper does not evaluate the impact of these trade-offs on control application objectives; our study aims to kick-start investigation into this area. Similarly, Hyperflow [13] is a distributed event-based control plane for OpenFlow that allows control applications to make decisions locally by passively synchronizing network-wide views of the individual controller instances. The paper evaluates the limits of how fast the individual controllers can synchronize and the resultant inconsistency, but does not evaluate the impact of this inconsistency on the application objective. Consistent Updates [10] focuses on state management between the physical network and the network information base (NIB) to enforce consistent forwarding state at different levels (per-packet, per-flow). It does not explore the implications of distributed SDN control plane state consistency on network objective performance.

Correctness vs. liveness trade-offs emerge in the context of current and historical intra- and inter-domain routing protocol design. Consensus Routing [6] presents a consistency-first approach to adopting forwarding updates in the context of inter-domain routing. Using distributed snapshots and a consensus protocol, each router ensures that every other router along the path toward a destination agrees on each routing update. The protocol separately addresses safety and liveness properties to achieve correctness guarantees on forwarding behavior, e.g. loop-free packet-forwarding. Various studies have examined the impact of stale and inaccurate intra-domain link state on quality-of-service oriented path selection objectives [12, 3]. Both measurement and analytical studies have characterized the effects of different link-state collection, aggregation, and update announcement patterns in terms of the resulting QoS path selection objectives.

Probabilistically Bounded Staleness [2] is a set of models that predicts the expected consistency of an eventually-consistent data store – the underpinning of an eventually-consistent distributed SDN control plane. It provides a useful platform for exploring design choice consequences and performance trade-offs for realizing replicated, distributed SDN control state.

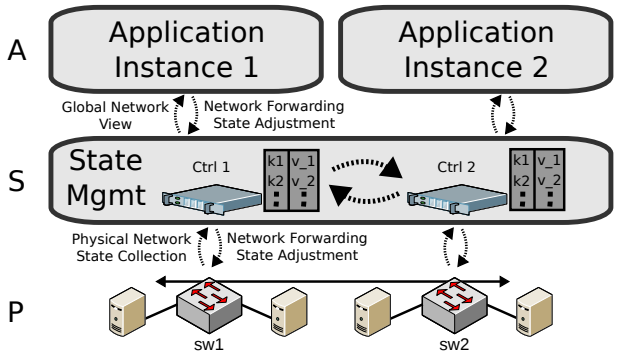


Figure 1: SDN state distribution and management conceptualized in layers: (A)pplication, (S)tate Management, (P)hysical Network

3. DISTRIBUTED STATE MANAGEMENT AND TRADE-OFFS

Before we dive into characterizing the state distribution and management trade-offs of SDN, we first describe the setting in which we define our problem. Figure 1 illustrates the key state exchange points in an SDN, organized into three logical layers. This representation is similar to SDN control platforms found in Onix [7]. Each dashed arrow in the figure indicates a state exchange point in the SDN.

At the bottom (layer P), the *physical network* consists of the hardware forwarding devices which store the *forwarding information base* (FIB) state of the network data plane (e.g., TCAM Entries and configured port speeds), as well as associated meta-data including packet, flow, and port counters. The devices of the physical network are grouped into one or more separate *controller domains*, where each domain has at least one physical *controller*. Figure 1 depicts two domains: "Ctrl 1" governs "sw 1", and "Ctrl 2" governs "sw2". The devices of each domain expose read and write interfaces for the meta-data and FIB state to the domain controller in the state management layer, indicated by layer S.

The state management (Layer S) is the core of the SDN, and is realized by the controllers of each domain, which collect the physical network state distributed across every control domain. This component is sometimes called the "Network Operating System" (NOS), as it enables the SDN to present an abstraction of the physical network state to an instance of the control application (Layer A), in the form of a global network view. The control logic for each application instance may be run as a separate process directly on the controller hardware within each domain.

Each controller maintains a *Network Information Base* (NIB) data structure, which is a view of the global network state presented to an application. For instance, the NIB contents presented to a network load-balancing SDN control application would include at least link capacity and utilization state. The NIB at each controller is periodically and independently updated with state collected from the physical network (e.g., through port counters or flow-level statistics gathering). Additionally, controllers synchronize their NIB state among themselves in order to disseminate their domain state to other controller domains.

Different manners of distributed, replicated storage models may

be used to realize the NOS state distribution and management, including transactional databases, distributed hash tables, and partial-quorum mechanisms [11]. One key property of any NOS state distribution approach is the degree of state consistency achieved – strong (e.g., via transactional storage) vs. eventual consistency. A strongly consistent NOS will never present inconsistent NIB state to an application (i.e., no two application instances will see a different global network view). However, the state distribution imposes overhead and thus limits the rate at which NOS state can be updated. While an update is being processed, applications continue to operate on a *stale* (but consistent) world view, even though more current information may be locally available. Eventually consistent approaches react faster, but temporarily introduce inconsistency – different global network views being presented to the individual physical controller instances.

Consequently, **trade-off #1** arises between the consistency model underlying NOS state distribution and the control application objective optimality. The performance of the network in relation to the control application's objective can suffer in the presence of inconsistent or stale global network view. Uncoordinated changes to the physical network state may result in routing loops, sub-optimal load-balancing, and other undesired application-specific behavior. The cost to achieving consistent state in the global network view entails higher rates of control synchronization and communication overhead, thus also imposing a penalty on responsiveness.

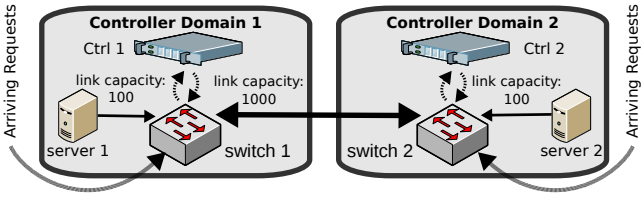
The degree to which the control application logic is more or less *aware* of the distributed nature of the underlying global network view constitutes **trade-off #2** between application logic complexity and robustness to stale NOS state. A "logically centralized" application that is unaware of the potential staleness of its input is simpler to design. An application which is aware of underlying distributed NOS state can take measures to separate and compare the inter-domain global network view with its own local domain view, and avoid taking action based solely on stale input.

4. EXAMPLE APPLICATION: NETWORK LOAD BALANCER

To investigate these trade-offs, we choose a well-known arrival-based network load balancer control application. The load balancer objective is to minimize the maximum link utilization in our network. We present and compare two implementations featuring different state (and staleness) awareness and management approaches.

Link Balancer Controller (LBC): The simpler of our two application approaches is inspired by Aster*x [5] and "Load Balancing Gone Wild" [14]. Within a specific domain, upon a dataplane-triggered event (e.g. reaction to a new flow arrival or pro-active notification of link imbalance within the domain), a global network view (table of links and utilizations) is presented by the NOS to the domain application instance. This view combines both the physical network state from within the domain as well as any inter-domain link utilization updates from other controllers. A list of paths (with utilizations) is generated from each ingress switch in the domain to every server which can respond to incoming requests (reachability information is provided by the NOS). From this list, the path with the lowest max link utilization is chosen on which to assign the next arriving flow and the appropriate forwarding state is installed in the physical network.

Separate State Link Balancer Controller (SSLBC): This control application keeps fresh intra-domain physical network state separate from updates learned through inter-domain controller synchronization events. The arrival-based path selection incorporates logic to ensure convergence properties on load distribution. Using the (potentially stale) global network view, it determines the ingress-to-server path P_{max}, with the maximum link utilization along the path. It calculates what fraction F of traffic *would* need to be redistributed off of P_{max}, (onto the other links) to better balance all the paths, however no active flows are actually migrated. Next, using only the fresh link utilization state from within its own domain, it calculates new link metrics within the domain. Using a convergence parameter $\alpha \in (0, 1]$, it scales each link utilization value by $F \times \alpha$ (a fraction of the domain's contributed load from path P_{max} across the links of other available paths). From this set of new link metrics, the path with the minimum max link utilization is then chosen for the next arriving flow. Effectively, the global network view guides each application instance to redistribute a scaled (α) fraction of its local link imbalance on a flow-by-flow arrival basis.



(a) Simulated topology

Link Name	Type	Capacity (Bandwidth)
s1-sw1	server-switch	100 units
s2-sw2	server-switch	100 units
sw1-sw2	inter-switch	1000 units
sw2-sw1	inter-switch	1000 units

(b) Simulated link capacities

Figure 2: Simulated topology and link capacities

5. EXPERIMENTS

We now discuss our investigation of the trade-offs described in the last sections. We describe the custom simulator we use, discuss our experiment setup, and describe and discuss our initial findings.

5.1 Simulation

To explore our state distribution trade-offs in a controlled and deterministic manner, we develop a custom simulator to implement the key state-exchange interfaces of an SDN, as introduced in Section 3. We opt for a custom flow-level simulation in this study, as we are first and foremost interested in having explicit control over the specific aforementioned state-exchange interfaces. We release our simulation as an open-source tool [1].

Our simulation is designed to capture interactions between three SDN layers from Figure 1. First, a graph data-structure [4] representing the physical network is instantiated with a topology structure, and link capacity and utilization annotations. Next, the NOS is instantiated from individual controller instances and controller domains are mapped to the network graph. Each controller is given its own copy of the graph data-structure as its NIB to update and distribute among the other controllers. Finally a flow workload definition is created, which is list of 4-tuples: Arrival time (in simulation time), ingress switch (where it enters our simulated network), duration, and average link utilization. The workload, controllers, and physical topology are provided to the simulation, and the simulation then begins iterating through the workload.

As the simulation iterates through the workload, simulation time is updated to the time of the flow arrival. Any existing flows which

have ended prior to this time are terminated, and their consumed link utilization is freed back into the available link capacity along the path it used. Each controller then records the link utilization values from the physical network within its domain. Next, an all-to-all synchronization may be triggered depending on whether the chosen synchronization period for the simulation run has elapsed.

The controller application of the domain in which the flow arrives is given the opportunity to assign the new flow to a path to any of the servers of which it is aware. Each controller application is able to assign flows entering from its domain's ingress switch to a server replica in the other controller domain. Based on its view of the network and the specific state management approach of the control application, the flow will be allocated to a path with the objective of minimizing the maximum link utilization in the network.

5.2 Experiment Setup

For our initial experiments, we choose our topology to be as simple as possible, yet still involve distributed state – two cooperating controller domains, as illustrated in Figure 2(a). Each domain consist of a single switch and a single server. For simulation purposes, we consider upstream traffic (e.g., http requests toward servers) negligible compared to downstream (e.g., streaming download content toward the switches), and therefore only simulate the link capacities and utilizations of four links in total, as given by the table in Figure 2(b). We consider both servers as identical replicas, able to serve content for all requests, constrained only by the available downstream bandwidth.

We choose fat inter-domain links, as they enable both domains to better cooperate in serving incoming requests. Alternately, a bottleneck between domains incentiveizes each domain to keep flows within its domain – limiting the value of an inter-domain cooperative load-balancing application.

The load balancer objective is to minimize the difference between all link utilizations, we choose RMSE – root mean squared error (the Euclidean distance) – of the maximum link utilization along each server-switch path in the network. Thus, if the maximum link utilization over every server-to-switch path is equal, our RMSE metric is 0.

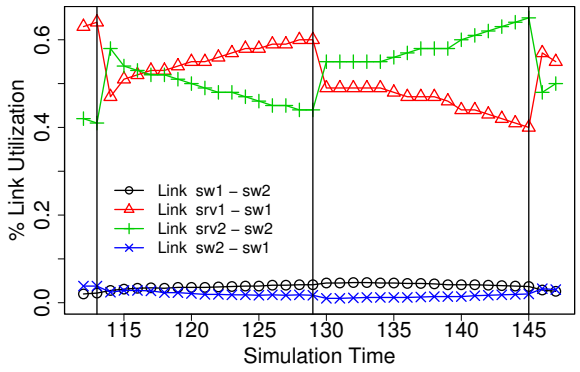
Although we present initial results only for this simple topology, simulations run on 3-domain chain and ring topologies exhibit very similar behavior.

5.3 Results

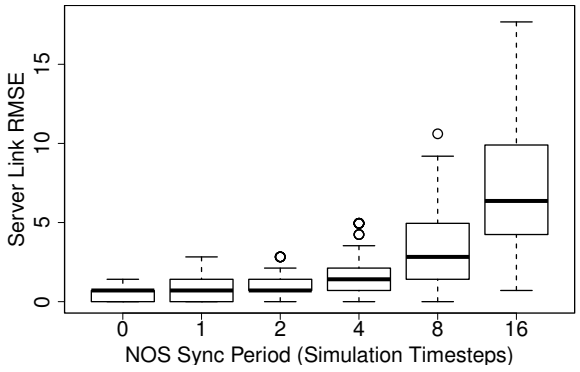
We use two different workloads to drive our simulations and explore the impact of inconsistent NOS state on network load balance. We first utilize a deterministic, controlled workload to impart a link utilization imbalance. This workload oscillates ingress load between the two switches, keeping constant the total load ingress into the network. Second, we apply a more realistic workload using exponentially distributed flow inter-arrival times and Weibull distributed flow durations.

5.3.1 Controlled workload—LBC

We realize the first of our two workloads, by choosing a flow arrival rate that is driven by a sin-function. More specifically, the flow arrival rate at each switch over a given simulation time interval $(t, t+1)$, is defined by $sin(t/T)$ and $sin(t/T - T/2)$ respectively, where T is the period of the oscillation. We choose a wave period of $T = 64$ simulation timesteps and run the simulation for 256 time steps, which leads to oscillating link utilization dynamics. We consider different workloads consisting of $32, 64, 128$ flows arriving within each time period to understand the impact of medium, heavy, and over-subscribed workloads respectively. We present results us-



(a) Link Utilization Timeseries (Sync Period = 16)



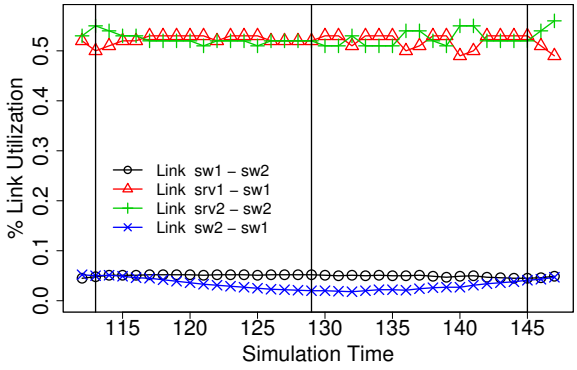
(b) Load Balancer Performance vs. Sync Period

Figure 3: LBC: global network view inconsistency vs. control application performance

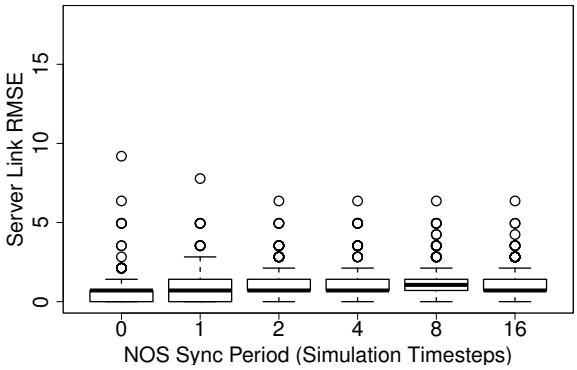
ing 32 arrivals per timestep with fixed flow duration of 2 timesteps. This gives a total ingress of 64 load units into the network at any point in time, keeping in mind that each server-to-switch link can carry a maximum capacity of 100 units.

We now use the load balancer based upon the LBC state management approach with different synchronization intervals: $0, 1, 2, 4, 8$ and 16 timesteps. Here, 0 implies that we synchronize the state between any two changes in NOS state. For the simulation with synchronization interval 16 Figure 3(a) shows an excerpt of the link utilization time series progression for the time period of 112 to 147. From the figure we confirm that the two server-switch links are indeed the "bottlenecks" with a utilization in the range of 40% to 60%. We also see the impact of synchronization at time steps 113, 129, and 145 (see support lines). After each synchronization step the load balancer reassigns a significant fraction of newly arriving flows to the other server and therefore to the other path — thus significantly changing the link utilizations. This results in an improved link balance for about 4-6 time steps later. Beyond that time, however, diverging link utilization state leads to inconsistent global network view as seen by each controller. The load balancer makes increasingly poorer decisions and the link imbalance grows. Also due to the oscillations in the workload, the switch with the greater ingress load changes over time. Similar patterns apply for shorter synchronization intervals and higher workload flow arrival rates.

To evaluate the load balancer performance, we consider the RMSE values over each simulation run, and exclude the first 16 timesteps. In Figure 3(b), we present the RMSE values for simulation run of increasing synchronization period in the form of a box-plot. Each

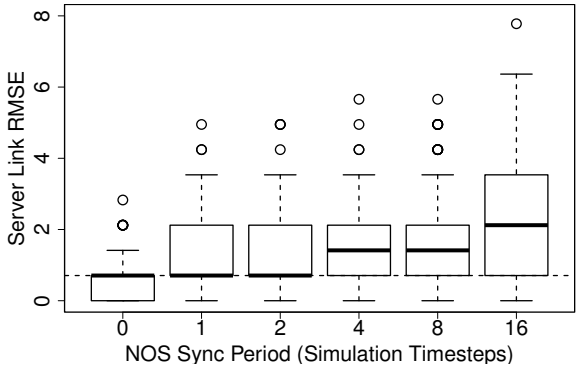


(a) Link Utilization Timeseries (Sync Period = 16)

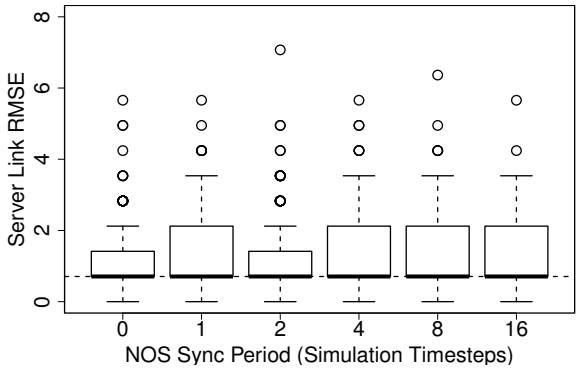


(b) Load Balancer Performance vs. Sync Period

Figure 4: SSLBC: global network view inconsistency vs. control application performance



(a) LBC Performance vs. Sync Period



(b) SSLBC Performance vs. Sync Period

Figure 5: Load balancer performance comparison under more realistic workload

box-plot shows the center half of the data (the box) with the median marked. The whiskers show the 95 percentiles and outliers are drawn separately. Recall, an RMSE of zero corresponds to balanced loads. We see that the RMSE value range increases as the synchronization periods increases. This is due to the effects highlighted by Figure 3(a). The median RMSE at sync period 16 is over $8\times$ the imbalance at sync period 1. This underlines our first trade-off – as the global network view becomes inconsistent, the application performance (in this case load balancing) can suffer.

5.3.2 *Controlled workload—SSLBC*

Next, we examine the second trade-off, namely, how the SSLBC state management approach is able to handle the above workload. Figure 4(b) shows the box-plot of the RMSE metric for each synchronization interval using the SSLBC state management approach. Comparing Figure 3(b) and 4(b) we observe that at high NOS synchronization rates, SSLBC achieves a mean RMSE comparable to LBC. As NOS staleness increases, the performance of SSLBC degrades far less as compared to LBC. However, for high rates of NOS synchronization, SSLBC may not perform as well because it is more conservative with respect to load balancing. Alternately, this improves SSLBC robustness to NOS staleness.

For example, Figure 4(a) shows the link utilizations for the same period as Figure 3(a). We see again see the effect of synchronization as well as the effect of the oscillations in the imposed load. However, the effects are much smaller, leading to lower RMSE values. Thus, we see the effects of the second trade-off – a more conservative control application design which is aware of underlying distributed state leads to less sensitivity to NOS staleness.

5.3.3 *Toward a more realistic workload*

We now evaluate the two previously illustrated trade-offs in the context of a more realistic workload. The simplistic sin function workload from earlier effectively illustrates the cost of NOS inconsistency to the control application performance. A more realistic flow arrival process and duration distribution is necessary, however, to better estimate how these expected trade-off may behave in practice.

This more realistic workload uses exponentially distributed flow inter-arrival times to define the ingress load on a per-switch basis. We modulate the mean of the exponential distribution by the wave function from earlier to achieve ingress load oscillations between switch 1 and switch 2. Flow durations are obtained from Weibull distribution with mean 10 (timesteps) and shape 0.5 to achieve a network ingress load over the simulation that is comparable to the earlier presented workload.

In this evaluation, as before, we vary the time interval between NOS synchronization events, and compare the load balancing performance of our LBC and SSLBC state management approaches. We reuse the same performance metric, RMSE over the server links and for each synchronization interval, present a box-plot of these values over an entire simulation run. We see in Figure 5(a) that as the sync period increases, the LBC again exhibits a noticeable increase in median RMSE. By comparison, Figure 5(b) shows the SSLBC performance remains almost unchanged for increasing sync periods. Additionally, at each synchronization period, the median RMSE of the SSLBC is equal to or strictly less (better) than that of the LBC application state management approach. The SSLBC shows less improvement over the LBC at lower synchro-

nization periods, as a heavy-tailed flow duration distribution can not be so easily accommodated by an arrival-based load balancing approach. These results support our earlier conclusions on the impact of stale NOS state on the control application performance.

6. SUMMARY

This position paper investigates the details behind the catchy tagline *Logically Centralized* of Software Defined Networks. Logically centralized world views, as presented by controller frameworks such as Onix [7] enable simplified programming models. However, as the logically centralized world view is mapped to a physically distributed system, fundamental trade-offs emerge that affect application performance, liveness, robustness, and correctness. These trade-offs, while well studied in a different context in the distributed systems community, are relevant to the networking community as well. It is our position that they should be revisited in the context of design choices exposed by software defined networks.

In this paper, we characterize the key state distribution points in the distributed SDN control plane. We identify two concrete trade-offs, between *staleness* and *optimality*, and between *application logic complexity* and *robustness to inconsistency*. We further discuss a simulation based approach to experimentally investigate these trade-offs. For a well-known SDN application (load balancing) and a simple topology, we find that *(i)* view staleness significantly impacts optimality and *(ii)* application robustness to inconsistency increases when the application logic is *aware* of distribution.

Although our current work focusses on a load-balancing control application scenario, note that similar trade-offs arise in other SDN control applications, e.g, in distributed firewalls, intrusion detection, admission enforcement (policy enforcement correctness vs. liveness), routing, and middle-box applications [9] (path choice optimality vs. liveness).

Certainly, more work is required to make our results quantitatively meaningful in the general SDN context. In future work, we

- plan to extend our *simulator* to more realistically model traffic characteristics and delays, and the overhead of synchronization
- apply our tools to larger and more complex *topologies*
- and compare the results of different *applications*.

Our hope is that this paper may spark a discussion about the details and trade-offs of logical centralization. Building on our early model and prototype, we aim to develop a tool-set that facilitates practical exploration of these state-distribution trade-offs for further applications, their concrete algorithms, workloads, and topologies.

7. ACKNOWLEDGEMENTS

We wish to extend gratitude toward Stefan Schmid for creative discussions as well as to our anonymous reviewers for their helpful feedback. This work was funded in part by Deutsche Telekom Innovation Laboratories, the GLAB project, and a DAAD research fellowship.

8. REFERENCES

[1] `github.com/cryptobanana/sdnctrlsim`.

[2] P. Bailis, S. Venkataraman, J. M. Hellerstein, M. Franklin, and I. Stoica. Probabilistically bounded staleness for practical partial quorums. Technical Report UCB/EECS-2012-4, EECS Department, University of California, Berkeley, Jan 2012.

[3] R. A. Guérin and A. Orda. QoS routing in networks with inaccurate information: theory and algorithms. *IEEE/ACM Trans. Netw.*, 7(3):350–364, June 1999.

[4] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *SciPy2008*, Aug. 2008.

[5] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari. Plug-n-Serve: Load-Balancing Web Traffic using OpenFlow. SigComm Demonstration, 2009.

[6] J. P. John, E. Katz-Bassett, A. Krishnamurthy, T. Anderson, and A. Venkataramani. Consensus routing: the internet as a distributed system. In *NSDI*, 2008.

[7] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A distributed control platform for large-scale production networks. In *USENIX OSDI*, 2010.

[8] 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 CCR*, 38(2), 2008.

[9] S. Raza, G. Huang, C.-N. Chuah, S. Seetharaman, and J. P. Singh. Measurouting: a framework for routing assisted traffic monitoring. *IEEE/ACM Trans. Netw.*, 20(1):45–56, Feb. 2012.

[10] M. Reitblatt, N. Foster, J. Rexford, and D. Walker. Consistent updates for software-defined networks: change you can believe in! In *ACM HotNets Workshop*, 2011.

[11] Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, Mar. 2005.

[12] A. Shaikh, J. Rexford, and K. G. Shin. Evaluating the impact of stale link state on quality-of-service routing. *IEEE/ACM Trans. Netw.*, 9(2):162–176, Apr. 2001.

[13] A. Tootoonchian and Y. Ganjali. Hyperflow: a distributed control plane for openflow. In *USENIX INM/WREN*, 2010.

[14] R. Wang, D. Butnariu, and J. Rexford. Openflow-based server load balancing gone wild. In *Proc. USENIX HotICE*, 2011.