# On Controller Performance in Software-Defined Networks

Amin Tootoonchian
*University of Toronto/ICSI*

Sergey Gorbunov
*University of Toronto*

Yashar Ganjali
*University of Toronto*

Martin Casado
*Nicira Networks*

Rob Sherwood
*Big Switch Networks*

## 1 Introduction

Network architectures in which the control plane is decoupled from the data plane have been growing in popularity. Among the main arguments for this approach is that it provides a more structured software environment for developing network-wide abstractions while potentially simplifying the data plane. As has been adopted elsewhere [11], we refer to this split architecture as Software-Defined Networking (SDN).

While it has been argued that SDN is suitable for some deployment environments (such as homes [17, 13], data centers [1], and the enterprise [5]), delegating control to a remote system has raised a number of questions on control-plane scaling implications of such an approach. Two of the most often voiced concerns are: *(a)* how fast can the controller respond to data path requests?; and *(b)* how many data path requests can it handle per second?

There are some references to the performance of SDN systems in the literature [16, 5, 3]. For example, an oft-cited study shows that a popular network controller (NOX) handles around 30k flow initiation events[1] per second while maintaining a sub-10ms flow install time [14].

Unfortunately, recent measurements of some deployment environments suggests that these numbers are far from sufficient. For example, Kandula *et al.* [9] found that a 1500-server cluster has a median flow arrival rate of 100k flows per second. Also, Benson *et al.* [2] show that a network with 100 switches can have spikes of 10M flows arrivals per second in the worst case. In addition, the 10ms flow setup delay of an SDN controller would add a 10% delay to the majority of flows (short-lived) in such a network.

This disconnect between relatively poor controller performance and high network demands has motivated a spate of recent work (*e.g.*, [6, 18]) to address perceived architectural inefficiencies. However, there really has been no in-depth study on the performance of a traditional SDN controller. Rather, most published results were gathered from systems that were never optimized for performance. To underscore this point, as we describe in more detail below, we were able to improve the performance of NOX, an open source controller for OpenFlow networks, by more than 30 times.

Therefore, the goal of this paper is to offer a better understanding of the controller performance in the SDN architecture. The specific contributions are:

**We present *NOX-MT* a publicly-available multi-threaded successor of NOX [8].** The purpose of NOX-MT is to establish a new lower bound on the maximum throughput. Unlike previous studies and implementations that were not tuned for performance, NOX-MT uses well-known optimization techniques (*e.g.*, I/O batching) to improve the baseline performance. These optimizations lets NOX-MT outperform NOX by a factor of 33 on a server with two quad-core 2GHz processors.

**We design a series of flow-based benchmarks** embodied in our tool, **cbench**, that we make freely available for others use. Cbench emulates any number of OpenFlow switches to measure different performance aspects of the controller including the minimum and maximum controller response time, maximum throughput, and the throughput and latency of the controller with a bounded number of packets on the fly.

**We present a study of SDN controller performance using four publicly-available OpenFlow controllers:** NOX, NOX-MT, Beacon, and Maestro [4]. We consider NOX as the baseline for our performance study since it has been previously used in different papers [14, 18, 6].

## 2 NOX-MT

NOX – whose measured performance motivated several recent proposals on improving control plane efficiency

[1]Throughout this paper we use flow initiation and requests interchangeably.

– has a very low flow setup throughput and large flow setup latency. Fortunately, this is not an intrinsic limitation of the SDN control plane: NOX is not optimized for performance and is single-threaded.

We present *NOX-MT*, a slightly modified multi-threaded successor of NOX, to show that with simple tweaks we were able to significantly improve NOX's throughput and response time. The techniques we used to optimize NOX are quite well-known including: I/O batching to minimize the overhead of I/O, porting the I/O handling harness to Boost Asynchronous I/O (ASIO) library (which simplifies multi-threaded operation), and using a fast multiprocessor-aware malloc implementation that scales well in a multi-core machine. Despite these modifications, NOX-MT is far from perfect. It does not address many of NOX's performance deficiencies, including but not limited to: heavy use of dynamic memory allocation and redundant memory copies on a per-request basis, and using locking were robust wait-free alternatives exist. Addressing these issues would significantly improve NOX's performance. However, they require fundamental changes to the NOX code base and we leave them to future work.

To the best of our knowledge, NOX-MT was the first effort in enhancing controller performance and motivated other controllers to improve. The experiments presented in this paper were performed in May 2011. Ever since all controllers studied in this paper have significantly changed and have different performance characteristics. We emphasize that our goal in this paper is to show that SDN controllers can be optimized to be very fast.

## 3 Experiment Setup

In an effort to quantify controller performance, we create a custom tool, *cbench* [12], to measure the number of flow setups per second that a controller can handle. In SDN, the OpenFlow *controller* [2] must setup and tear down flow-level forwarding state in OpenFlow switches. This "flow setup" process can happen statically before packets arrive ("proactively") or dynamically as part of the next hop lookup process ("reactively"). Reactive flow setups are particularly sensitive because they add latency to the first packet in a flow. Once set up, the flow forwarding state remains cached on the OpenFlow switch so that this process is not repeated for subsequent packets in the same flow. The OpenFlow controller also communicates how long to cache the state: either indefinitely, after a fixed timeout, or after a period of inactivity. We choose to focus on the flow setup process because both because it is integral to SDN and because it is perceived to be the likeliest source of performance bottleneck.

Our tool, cbench [12], measures various performance issues related to flow setup time. Cbench emulates a configurable number of OpenFlow switches that all communicate with a single OpenFlow controller. Each emulated switch sends a configurable number of new flow (OpenFlow *packet_in*) messages to the OpenFlow controller, waits for the appropriate flow setup (OpenFlow *flow_mod* or *packet_out*) responses, and records the difference in time between request and response.

Cbench supports two modes of operation: latency and throughput mode. In latency mode, each emulated switch maintains exactly one outstanding new flow request, waiting for a response before soliciting the next request. Latency mode measures the OpenFlow controller's request processing time under low-load conditions. By contrast, in throughput mode, each switch maintains as many outstanding requests as buffering will allow, that is, until the local TCP send buffer blocks. Thus, throughput mode measures the maximum flow setup rate that a controller can maintain. Cbench also supports a hybrid mode with *n*-new flow requests outstanding, to explore between these two extremes.

We analyzed cbench to ensure that it is not a bottleneck in our experiments. For that, we instrumented cbench to report the average number of requests on the fly for different experiments. Cbench also reports average throughput and response time. According to Little's theorem ($L = \lambda W$) [10] the average number of outstanding requests must match the product of the system throughput and the average response time. Throughout our experiments these numbers were in agreement. The slight differences are an artifact of taking the average over the samples collected in each run.

Using cbench, we evaluated the flow setup throughput and latency of four publicly available OpenFlow controllers using *cbench*: *(a)* NOX [8] is a single-threaded C++ OpenFlow controller adopted by both industry and academia. *(b)* NOX-MT is an optimized multi-threaded successor of NOX we developed and presented in this paper. *(c)* Maestro [4] is a multi-threaded Java-based controller from Rice university. *(d)* Beacon[3] is a multi-threaded Java-based controller from Stanford university and Big Switch Networks. We used the latest available version of each controller available as of May 2011.

In all the experiments, each controller runs the L2 switching application provided by the controller.[4] For

[2]While the specifics of this description use an OpenFlow-based terminology, the flow setup operation is common to all flow-based network architectures, e.g., setting up virtual circuits in ATM, configuring labels in MPLS, or physical circuits in optical networks.

[3]Through private correspondence with Beacon's author, we were informed that Beacon performs twice better on a 64-bit OS. We note that the results reported in this paper are for 32-bit runs. However, we are not aware of the reason for this gap in performance.

[4]We choose to use L2 switching for our evaluation for two primary reasons. First, it provides a lower-bound for lookup since the full forwarding decision can be accomplished with a hash followed by a sin-

each switch on the path, the switch application performs MAC address learning. Each packet is forwarded out of the last port on which the traffic from the destination MAC address is seen. Packets with unknown destinations are flooded. In NOX, for each switch, the mapping between the MAC-switch tuple and the port number is stored in a hash table. The switch application has mostly read-only work load: only requests with newly observed source MAC addresses trigger an insert/update in the hash table, and the number of such events is proportional to the product of the number of hosts in the network and the number of switches.

To minimize interference, we run the controller and cbench on two separate servers ($8 \times 2GHz$ and $4 \times 2.13GHz$ CPU cores respectively, both with 4GB of DDR2 ram)[5]. Each server has two Gigabit ports directly connected to the other server. The Gigabit links are teamed together to provide a 2Gbps control bandwidth required for some experiments. Throughout our experiments cbench's CPU utilization is consistently less than $50\%$. We occasionally run multiple parallel instances of cbench on different processors to verify cbench's fairness in serving different emulated switches (sockets) as well as its accuracy.

Each test consists of 4 loops each lasting 5 seconds. The first five seconds (first loop) is considered as controller warm-up and its results are discarded. Each test uses 100k unique MAC addresses (representing 100k emulated end hosts). For experiments with fixed number of switches, we chose to present the results for 32 emulated switches because we do not expect a large number of switches to be stressing the network simultaneously. To maximize the stress on the controller and to ensure that the bandwidth is not the bottleneck to the extent possible, we use 82-byte sized OpenFlow packet-in messages (referred to as requests in the graphs).[6]

gle lookup. Thus, the test is not heavily augmented by overhead of the lookup (which could be the case with something like longest-prefix match implemented in software). And secondly, basic switching has been implemented in all the controllers we tested.

[5]All servers run Debian Squeeze with Linux `2.6.32-5-686-bigmem`, gcc `4.4.5`, GNU libc `2.11`, GNU libC++ `3`, Boost `1.42`, Sun Java `1.6`, and TCMalloc `1.5`. NOX and NOX-MT are compiled with no support for python and with debugging disabled. All controllers are run with logging disabled and no verbose output. For NOX, NOX-MT, and Maestro threads are bound to distinct CPUs. We used TCMalloc [7] malloc implementation since it provides a faster alternative to GNU libc's malloc with better scalability in multi-threaded programs. Also, unless otherwise noted, we run the experiments with default sizes for Linux networking stack buffers as well as NIC drivers' ring buffers. Cubic is the default congestion control algorithm on Linux 2.6.32.

[6]All our estimates on the bandwidth usage is based on this size. For instance, throughout the paper we map 1.45M requests to 1Gbps control bandwidth. That is because $1.45\frac{Mreq}{sec} \times 82\frac{bytes}{req} \times 8\frac{bits}{byte} = 951.2Mbps$ and assuming that the majority of Ethernet frames are MTU-sized (to account for Ethernet+IP+TCP overhead), this number roughly corresponds to 1Gbps.

## 4 Controller Throughput

Individual controllers' throughput is an important factor in deciding the overall number of controllers required to handle the network control load. Our focus in this section is to study the maximum throughput in the system in various settings.

**Maximum throughput:**
Figure 1 shows the average maximum throughput of different controllers with different number of threads. The results suggest that: *(a)* NOX-MT shows a significantly better performance compared to the other controllers. It saturates 1Gbps control bandwidth with four 2GHz CPU cores. *(b)* As expected, all the multi-threaded controllers achieve near-linear scalability with the number of threads (cores), because the controller's workload is mostly read-only minimizing the amount of required serialization.

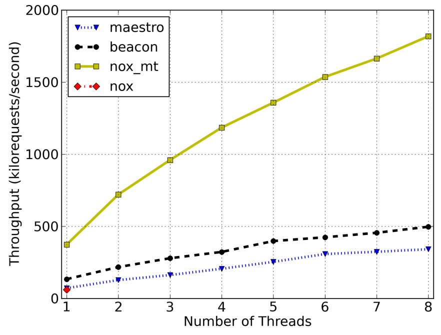


**Figure 1:** *Average maximum throughput achieved with different number of threads.* We use 32 emulated switches and 100k unique MAC addresses per switch. NOX-MT saturates 1Gbps (1.45Mreq/sec) of control bandwidth using four 2GHz CPU cores. Since NOX is single-threaded it only has a single point in this and similar graphs.

**Relation with the number of active switches:**
Ideally, controller's aggregate throughput should not be affected by the number of switches connected to it. However, increased contention across threads, TCP dynamics, and task scheduling overhead within the controller are factors that can lead to a degraded performance if we have a large number of highly active switches.

To study the impact of the number of switches on controller performance, we measure the average maximum throughput with different number of switches and threads in Figure 2. We observe that, adding more threads beyond the number of active switches does not improve throughput. Also, increase in the number of switches beyond a threshold reduces the overall throughput, because: *(a)* I/O handling overhead increases, *(b)* contention on the task queue and other shared resources increases, and *(c)* I/O and job batching become less effective. We note that the number of switches presented in our experiments only reflects the highly active (i.e. producing an extremely large number of requests per second) switches. In a typical network it is very unlikely that all the switches are highly active at the same time, so each controller should be able to manage a far larger number of switches.

**Relation with the load level:**
Cbench's throughput mode effectively keeps the pipe between itself and the controller full all the time. However, since we have large buffers across all layers in the networking stack (*e.g.*, network adapter's ring buffer, network interface's send and receive queues, TCP buffers, etc.), this pipe is quite large.

Our experiments verify that average maximum throughputs with $2^{12}$ outstanding requests are very close to the ones with unlimited number of outstanding requests (see Figure 3). With the same throughput, it follows from the Little's law that doubling the number of requests in the system doubles the response time. In our experiments, the average delay with unlimited outstanding requests is almost an order of magnitude larger than the limited ones even though they achieve the same level of throughput. We study the controller response time corresponding to different load levels in Section 5.

**Effect of write-intensive workload:**
Write-intensive workloads increase the contention in the network control applications. For the switch control application, having a large number of unique source MAC addresses result in a write-intensive workload. As shown in Figure 4, both Maestro and Beacon are significantly affected by this workload. However, NOX-MT does not exhibit a similar behavior. That is because NOX-MT's switch application minimizes contention in such scenarios by partitioning the network's MAC address table among a pool of hash tables selected by the hash of the MAC address.

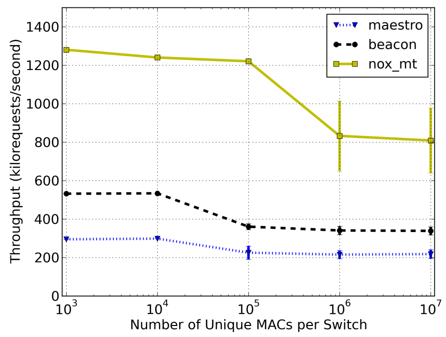


**Figure 4:** *Average maximum throughput for various levels of write-intensive workloads with 32 switches and 4 threads.*

## 5 Controller Response Time

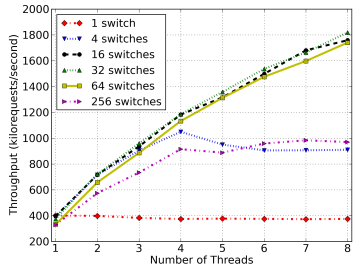
In a software-defined network where flow setup is performed *reactively*, controller response time directly affects the flow completion times. In this section, we present benchmarks to measure minimum (least load) and maximum controller (maximum load) response time of SDN controllers. Then, we study the relation between the load level and response time as well as the number of switches and response time.

**Minimum response time:**
To measure minimum control plane response time, we constrain the number of packets on the fly to be exactly one. Besides controller service time, this minimum response time includes traversal of networking stacks on both the controller and cbench sides twice, as well as the processing time of the controller. Average response times of all controllers are between 100 and 150 microseconds.

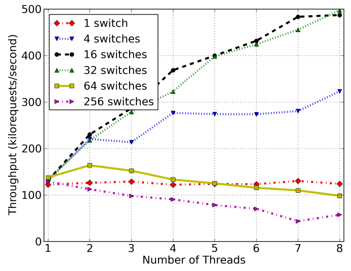
**Maximum response time:**
Maximum response time for each controller is observed when the maximum number of packets on the fly is not bounded (*i.e.*, when the benchmakrer exhausts all the buffers in between the emulated switch and the controller). Since NOX-MT has the highest throughput, it has the least response time compared to others. As we see in Table 1, the average number of packets on the fly across these experiments is inversely proportional to the controller throughput (in accordance with the Little's law)

**Relation with the load level:**
To better understand the relation between controller load and response time, we plotted the response time CDFs fixing the load level to $2^{12}$ requests on the fly (see Figure 5) and the response time varying the load level (see Figure 6). We find that for the same workload adding more threads decreases the response time. Also, doubling the number of outstanding requests doubles the response time, but does not significantly affect the throughput (see Section 4).

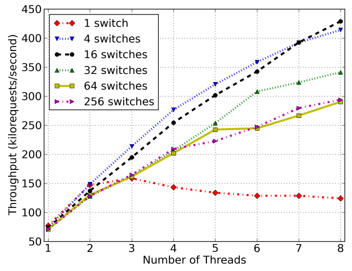
**Relation with the number of active switches:**
Varying the number of active switches (see Table 2), we observe the same pattern for delay as we observed for throughput in Figure 2. Adding more CPUs beyond the number of switches does not improve latency, and serving far larger number of switches than available CPUs results in a noticeable increase in the response time.



**(a)** NOX-MT

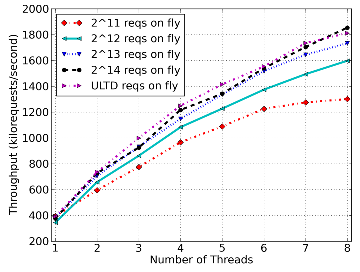


**(b)** Beacon

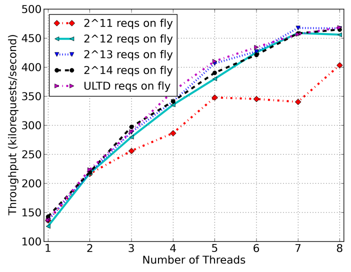


**(c)** Maestro

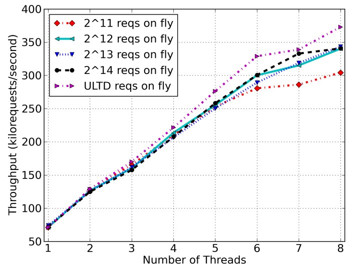
**Figure 2:** *Average maximum throughput with different number of switches.* NOX-MT shows nearly identical average maximum throughput for 16, 32 and 64 emulated switches. With 256 emulated switches the performance of all three controllers degrades.



**(a)** NOX-MT



**(b)** Beacon

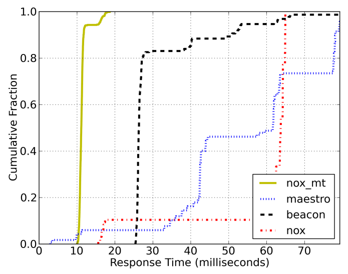


**(c)** Maestro

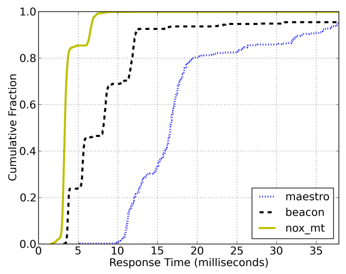
**Figure 3:** *Average maximum throughput with different number of threads and limits on the maximum overall number of requests on the fly.* It is possible to achieve an almost maximum throughput even with limited number of requests on the fly for all three controllers.

| | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| NOX | $631.87 \pm 44.62$ | - | - | - |
| NOX-MT | $349.44 \pm 127.45$ | $143.55 \pm 63.19$ | $92.59 \pm 42.40$ | $66.34 \pm 38.32$ |
| Beacon | $1028.51 \pm 175.32$ | $634.83 \pm 204.99$ | $394.21 \pm 205.59$ | $293.80 \pm 233.33$ |
| Meastro | $1268.58 \pm 84.38$ | $783.56 \pm 56.56$ | $558.40 \pm 338.04$ | $361.01 \pm 301.68$ |

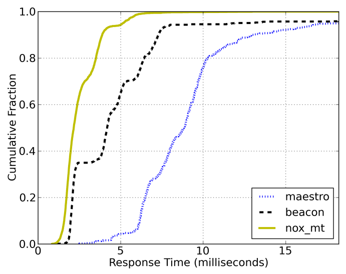
**Table 1:** *Worst-case average response time and standard deviation (milliseconds) for various number of threads with 32 switches and unlimited number of outstanding flow setup requests.*



**(a)** Single-threaded



**(b)** Four threads

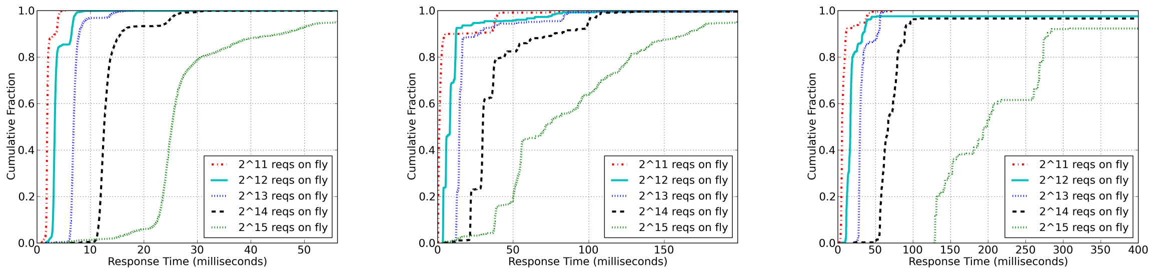


**(c)** Eight threads

**Figure 5:** *Response time CDF for different controllers with 32 switches, and $2^{12}$ maximum requests on the fly.* NOX-MT has the lowest response time for 1, 4 and 8 threads.

| | 1 | 4 | 16 | 32 | 64 | 256 |
|---|---|---|---|---|---|---|
| NOX-MT | $9.92 \pm 6.39$ | $3.80 \pm 0.69$ | $3.51 \pm 1.10$ | $3.84 \pm 2.51$ | $4.63 \pm 6.65$ | $8.63 \pm 18.73$ |
| Beacon | $30.47 \pm 14.99$ | $14.85 \pm 18.69$ | $11.89 \pm 26.22$ | $11.86 \pm 35.69$ | $18.74 \pm 85.57$ | $30.48 \pm 116.02$ |
| Meastro | $22.75 \pm 10.55$ | $15.98 \pm 10.18$ | $15.97 \pm 11.08$ | $26.09 \pm 43.76$ | $23.35 \pm 49.69$ | $29.84 \pm 57.88$ |

**Table 2:** *Response time (milliseconds) varying the number of switches for runs with 4 threads and $2^{12}$ requests on the fly.*



**(a)** NOX-MT **(b)** Beacon **(c)** Maestro

**Figure 6:** *Response time CDF for various maximum number of requests on the fly with 32 switches, and four threads.*

## 6 Concluding Remarks

We present NOX-MT that establishes a new lower bound on the maximum throughput for SDN controllers. Our microbenchmarks demonstrate that existing controllers all perform significantly better than what current literature assumes. On an eight-core machine with 2GHz CPUs, NOX-MT handles 1.6 million requests per second with an average response time of 2ms. We emphasize that controller responsiveness is the primary factor to decide if additional controllers should be deployed.

We are not, however, suggesting that a single *physical* controller is enough to manage a sizeable network. High availability and maintaining low response times are among the reasons that a network needs multiple controllers. However, given the low frequency of failures and changes to the network topology, it seems that maintaining a consistent logically centralized view of the network across controllers (*e.g.*, [15]) is feasible.

Finally, we note that understanding overall SDN performance remains an open research problem. System-wide performance is likely a complex function of topology, work load, equipment, and forwarding complexity. Our single-controller microbenchmarks presented in this paper are just a first step towards the understanding of the performance implications of SDN.

## References

[1] Al-Fares, M., Loukissas, A., and Vahdat, A. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication* (2008), ACM.

[2] Benson, T., Akella, A., and Maltz, D. A. Network traffic characteristics of data centers in the wild. In *Proceedings of IMC 2010* (2010), ACM.

[3] Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A., and van der Merwe, J. Design and implementation of a routing control platform. In *Proceedings of the USENIX NSDI 2005* (Berkeley, CA, USA, 2005), USENIX Association.

[4] Cai, Z., Cox, A. L., and Ng, T. S. E. Maestro: A system for scalable OpenFlow control. Tech. Rep. TR10-11, Rice University - Department of Computer Science, December 2010.

[5] Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. Ethane: taking control of the enterprise. *SIGCOMM CCR 37*, 4 (2007), 1–12.

[6] Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. Devoflow: scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011* (2011), ACM.

[7] Google. Thread-Caching Malloc. `http://goog-perftools.sourceforge.net/doc/tcmalloc.html`.

[8] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. NOX: towards an operating system for networks. *SIGCOMM CCR 38*, 3 (2008), 105–110.

[9] Kandula, S., Sengupta, S., Greenberg, A., Patel, P., and Chaiken, R. The nature of data center traffic: measurements & analysis. In *Proceedings of IMC 2009* (2009), ACM, pp. 202–208.

[10] Leon-Garcia, A., Probability, and random processes for electrical engineering. *Probability, statistics, and random processes for electrical engineering*, 3rd ed. ed. Pearson/Prentice Hall, Upper Saddle River, NJ, 2009.

[11] `www.opennetworkingfoundation.org`.

[12] Rob Sherwood and Kok-Kiong Yap. Cbench: an Open-Flow Controller Benchmarker. `http://www.openflow.org/wk/index.php/Oflops`.

[13] Sundaresan, S., de Donato, W., Feamster, N., Teixeira, R., Crawford, S., and Pescapè, A. Broadband internet performance: a view from the gateway. In *Proceedings of the ACM SIGCOMM 2011* (Aug. 2011), ACM.

[14] Tavakoli, A., Casado, M., Koponen, T., and Shenker, S. Applying nox to the datacenter. In *Proceedings of HotNets-VIII* (2009).

[15] Tootoonchian, A., and Ganjali, Y. Hyperflow: a distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking* (Berkeley, CA, USA, 2010), USENIX Association, pp. 3–3.

[16] Yan, H., Maltz, D. A., Ng, T. S. E., Gogineni, H., Zhang, H., and Cai, Z. Tesseract: A 4D network control plane. In *Proceedings of the USENIX NSDI 2007* (2007).

[17] Yiakoumis, Y., Yap, K.-K., Katti, S., Parulkar, G., and McKeown, N. Slicing home networks. In *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks* (2011), ACM.

[18] Yu, M., Rexford, J., Freedman, M. J., and Wang, J. Scalable flow-based networking with difane. In *Proceedings of the ACM SIGCOMM 2010* (2010), ACM, pp. 351–362.