Challenges in the Emulation of Large Scale Software Defined Networks

Arjun Roy, Kenneth Yocum, and Alex C. Snoeren

University of California, San Diego

Abstract

Researchers and practitioners alike have long relied upon emulation to evaluate the performance of networked applications and protocols without tying up production infrastructure or committing to expensive hardware purchases. As software defined networks (SDN) become more prevalent, it is increasingly useful to be able to accurately emulate their behavior. Tools like Mininet-HiFi have demonstrated the viability of small-scale SDN emulation, but emulated network size is limited by the hardware resources of a single machine. Unfortunately, there are many SDN applications that can only be evaluated meaningfully at scale.

Traditionally, researchers have overcome similar challenges by scaling the capacity of the network links, the emulated workload, or both. We observe that shrinking a network arbitrarily to fit into a fixed set of emulation resources has fundamental limitations that impact the fidelity of results, especially in the context of SDN. In particular, we demonstrate cases where emulation yields false positives—poor network behavior that manifests only in emulation—and false negatives, where a real deployment would suffer issues not apparent in emulation. We discuss strategies that might be used to combat these effects and their limitations.

1 Introduction

Software defined networking is gaining traction in production networks due in part to its ability to allow operators to optimize for desired performance characteristics. To realize gains, however, operators may need to employ complex custom controller software. Hence, cautious operators will want to test their configurations on real workloads before deployment. Traditionally, emulation has been a key tool for such testing.

While existing SDN emulation tools are undeniably useful, we argue that they are ill-suited to evaluate large scale SDN deployments found in many production enterprise networks and data centers. In particular, there exist important SDN experiments that fundamentally require emulating the full breadth of the production network. Real deployments can feature thousands of nodes with complex distributed network controllers, introducing challenges that simply cannot manifest at small scale. However, emulating large networks poses significant technical challenges in the implementation of the emulation system itself.

In this paper, we examine the suitability of the traditional approach of "shrinking" a large software defined network to fit within a restricted resource budget by scaling down link bandwidths [9]. Shrinking is an attractive option, since it enables emulating large networks on a single emulator machine, reducing cost, implementation complexity, and the barrier to entry. We observe, however, that there are fundamental issues specific to SDN emulation that limit this approach. Using experiments representative of real-world application patterns, we show how scaled-down emulation introduces artifacts that cause certain SDN operations to be improperly modeled, and how these artifacts can mislead an operator by providing both false positive and false negative predictions of poor network behavior.

In the rest of this paper, Section 2 describes related work in the field of network emulation, within and outside the realm of SDN. Section 3 motivates the need for emulating large networks. Section 4 describes the issues present in shrinking emulated SDNs. Section 5 describes experiments justifying our arguments. Section 6 discusses strategies to solve these issues and limitations that they face. Finally, we conclude in Section 7.

2 Related work

Mininet-HiFi [5] is one of the first emulators developed explicitly to support SDN. It targets network-limited experiments, on topologies where all virtual client machines and switches can be emulated with the processing capacity of a single server. This configuration enables it to effectively run small-scale networks with artificial traffic. For example, the authors use it to demonstrate the behavior of Hedera [2] on a mix of elephant and regular flows, but it is not intended to run a heavy duty application like MapReduce at scale. In particular, the authors observe that "it is a challenge to reproduce results that depend on the absolute value of link/CPU bandwidth."

Modelnet [11], on the other hand, provides the ability to run unmodified applications over an emulated wide-area network. Its goal is not to reduce the amount of resources required to emulate a network, but, rather to reproduce the link properties (bandwidth, latency, etc.) of a target network. It handles the issue of scaling the size of the emulated network by partitioning the network over multiple emulator nodes. It does not, however, explicitly support the emulation of SDN hardware.

SHRiNK [9] provided some of the first evidence that an IP network can be scaled down using flow sampling and still provide accurate performance metrics on a network that carries a mix of long and short lived TCP-like and UDP flows. However, by ignoring end-system aspects, such an approach can yield poor approximations of performance for complex applications like MapReduce. Time dilation [4], however, showed how to reduce the resources required to faithfully emulate real applications by scaling a virtual machine's apparent performance, thereby making the network appear to run faster, or, if desired, at the same rate while using fewer physical resources like bandwidth.

DieCast [3] employs this latter time dilation use case to emulate complex, multi-tier services with limited physical resources. Researchers can set the relative amount of perceived resources dedicated to end-system (CPU and disk) and to network (link capacity) emulation. While this approach avoids the issue of interpreting down-sized experiments, DieCast does not consider the impact of time dilation on SDN switches or controllers.

3 Large network emulation

In the realm of SDN, the control and data planes of a forwarding element are separated. The control plane is remotely programmable by the network operator. Network behavior depends on this state, which can be distributed over the network. In Openflow, the state consists of the switch flow table and soft state stored on the switch controller. In large deployments, the controller may implemented as a distributed application for performance reasons [7].

Consequently, accurately modeling the behavior of the switch-to-controller control path is critical to determining how a network will perform [6]. In addition to straightforward metrics like network latency, the details of controller state distribution can impact performance as well [7]. Just as the authors of Mininet-HiFi call out the relative CPU-to-link bandwidth ratio as being important for accurate emulation, we observe that the SDN control plane must be appropriately scaled as well. This requirement complicates controller application validation, since emulations of small networks might mask effects that would be present in a large deployment.

For example, updating many switches with forwarding rules can lead to unpredictable ordering of update completion times, making it hard to faithfully emulate how packets would be routed in practice [8]. In corner cases, a particular ordering can lead to routing loops or black holes. Synchronizing state takes time, and a larger network has more to synchronize. Hence, distributed controller emulation on a small network is unlikely to provide results representative of larger deployments, since the maximum deviation from a synchronized network state is smaller than on a large network. Thus, when choosing controller placement or estimating performance, we need to emulate networks similar in size to the target network.

Further, large networks are needed to characterize the impacts of equipment failure, since they have non-trivial failure rates that are difficult to mimic in small networks. While, in general, failures can be injected into small deployments, any given failure will have a larger and possibly disproportionately severe impact. For example, a link failure might cause bulk flows to shift onto a link with synchronized RPC traffic, leading to latency spikes that might not have occurred in a larger network with more alternate forwarding paths. SDN further complicates the problem: Controllers can have arbitrary logic to deal with failures, and it can be difficult to characterize recovery performance when the emulated network is smaller than the target network.

4 Limitations on shrinking

The performance of an emulator is ultimately limited by the rate at which it can forward packets. An emulator with a fixed forwarding capacity must scale down link bandwidths in order to accommodate increasing scale in the emulated network. This trade-off is often acceptable; for example, long-lived TCP flows will fairly compete on low and high bandwidth links. The Hedera experiment re-creation by the authors of Mininet-HiFi uses 10-Mbps links, compared to 1-Gbps in the original experiment. Despite this, the emulated results follow the same trends when normalized. In contrast, we find that shrinking link speeds can trigger certain inherent artifacts of SDN emulation that cause a significant mismatch between real-world and emulated behavior.

In this section we discuss two such fundamental artifacts and demonstrate their particular impact on SDN networks in Section 5. In particular, we focus on control planes that are constrained by the amount of space available in forwarding elements to store Openflow rules, which is a major limiting factor across a variety of network control schemes.

4.1 The Clogged Drain effect

Shrinking link bandwidth increases flow completion time. However, as shown in Section 5, flow initiation rate does not necessarily decrease as quickly as link bandwidth. Suppose a switch services many small, independent flows, as in the case where users of a Web service simultaneously make small requests. Assuming a fixed rate of flow arrivals, more flows will coexist and compete for switch resources with low-bandwidth links than with faster links, since flow termination depends on throughput. The effect is reminiscent of a faucet over a clogged drain, with new flows arriving faster than existing flows can be drained.

This effect is important because a switch stores limited forwarding state. For example, the NEC PF5820 is limited to 750 Openflow entries [1]. When a flow arrives at a switch, if no entry matches, the switch must contact the controller for a new entry. If the flow table was full, the new entry might evict an existing entry. Openflow allows actions beyond forwarding. With a large number of potential actions for a large number of flows, we risk churn in the switch's flow table. While a switch might support a much larger exact match table for flows, this table is typically used to support non-programmable L2 switching, and thus does not provide the flexibility that characterizes SDN.

Bandwidth (Mbps)	Median 60% variance (seconds)
1000	0.0818
100	1.1627
10	3.2863
1	30.6746

Table 1: The time between the completion of the first 20% of 1,000 flows and the first 80% (i.e., the middle 60% of the flows) for four different link speeds.

4.2 The Straggling Herd effect

As link bandwidth decreases, so does the number of flows transmitting in a fixed period. For example, a link will transfer 89,479 packets/second at 1 Gbps and 1500-byte MTU, but only 87 packets/second at 1 Mbps. On a low bandwidth link with many flows, a majority of the flows will be unable to transmit packets at all in a given period. Consider a set of simultaneously starting small flows of similar size. We would expect them to terminate all at once or relatively soon thereafter. With a sufficiently low bandwidth link, they will be unable to do so. They will instead form a straggling herd, where flows that should complete at the same time are serialized and finish in distinct batches. Table 1 shows the extent to which this effect manifests for 1000 concurrent flows. At 1 Gbps, the bulk of the flows finish within 80 ms of each other; at 1 Mbps, the same fraction are spread apart by more than 30 seconds.

5 Demonstration

To demonstrate the impact of these artifacts, we use a modified Modelnet emulator, which interfaces with Open vSwitch to provide an emulated SDN in a manner similar to Mininet-HiFi. All tests are run on a single system with an Intel i7-950 processor and 12 GB of RAM, on Linux 3.2. Link bandwidth is controlled via Linux Traffic Control (`tc`). CPU allocations for emulated switches are controlled by Linux CFS bandwidth control as noted. Switches connect to a POX controller out of band of the emulated network. We use a reactive control plane as a stand-in for various interesting types of network management schemes that react to current network state.

5.1 Clogged Drain and false positives

We induce the Clogged Drain effect on a one-switch topology to demonstrate that low link bandwidth leads to flow table churn that does not exist at scale. To simulate shrinking a large network, we limit switch CPU allocation by a shrinking ratio. For example, a ratio of 10 corresponds to the case of 10 emulated switches, each receiv-

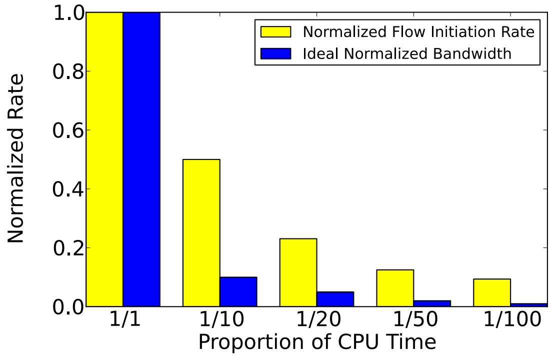

Figure 1: The rate of flow initiation and emulated bandwidth, both normalized to the 1:1 (non-scaled) case.

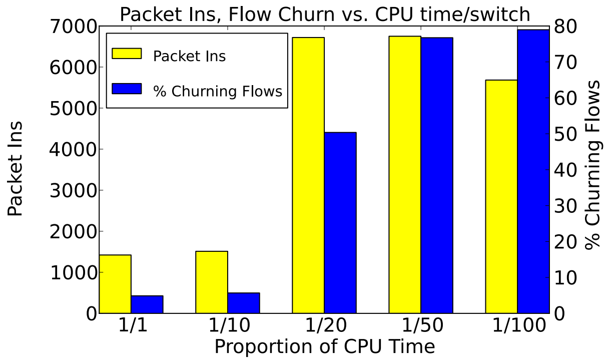

Figure 2: Packet-in events and flow churn for various scaling factors. There are more flows (1,236) than can fit in the flow table simultaneously (750).

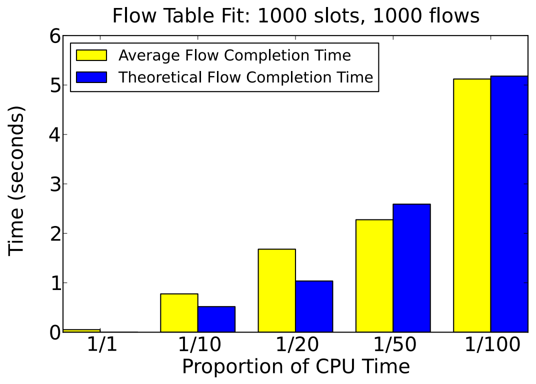

Figure 3: Completion times with no flow table contention.

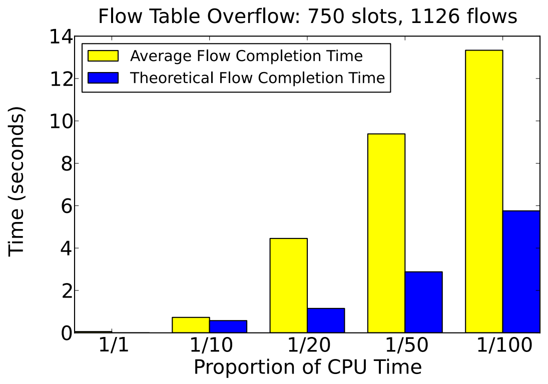

Figure 4: Completion times with flow table contention.

ing 1/10th CPU time. We start a fixed set of flows simultaneously and measure completion time and the number of "packet in" controller events. We use the term *stable* flows to refer to flows that cause one packet-in event, while *churning* flows cause more than one.

We investigate two cases: when all flows fit in the table (no contention), and when the number of flows is 1.5× flow-table capacity (contention for flow table space). In the first case, there are 1,000 flows with a switch flow table size of 1,000 (i.e., all flows should fit within the table). In the second case, there are 1,126 flows, with a flow table size of 750. We run the experiment with the switch receiving all, 1/10th, 1/20th, 1/50th and 1/100th of the run time of a single CPU, corresponding to the case of emulating 1, 10, 20, 50 and 100 switches. Links are not explicitly controlled by `tc`, but are limited by the available CPU on the switch.

Figure 1 tracks the observed normalized rate of flow initiation and the theoretical normalized aggregate bandwidth of the switch as a function of CPU time. If a switch receives $1/R$ of total CPU time, bandwidth must also scale by a factor of $1/R$. Each observed value of flow initiation rate is normalized to the rate observed in the single switch emulated case, which is approximately 187 flows initiating per second. As the proportion of switch CPU time goes down, flow initiation rate decreases, but, critically, not as fast as bandwidth decreases, leading to the Clogged Drain effect.

Convinced that the Clogged Drain effect is occurring in this experiment, we now turn to understanding its impact. Figure 2 plots the switch flow table churn for the second instance of the experiment, i.e., the one where the flow table cannot hold all the flows simultaneously. Both the number of packet-in events and the number of churned flows increases greatly as the CPU allocation for the switch diminishes, as predicted earlier. One might argue, however, that such SDN-specific details are immaterial to the goal of the emulation.

In contrast, Figures 3 and 4 show flow completion time (i.e., an end-to-end performance metric one would like an emulator to faithfully reproduce) in the absence and presence of flow table contention. In each case, we first

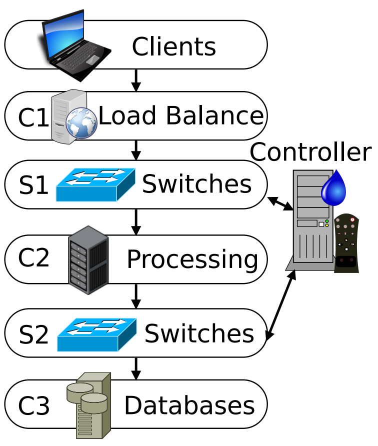

Figure 5: An example multi-tier Web service.

measure average flow completion time when the switch receives all available CPU time. Since bandwidth drops with the shrinking ratio R, flow completion time should rise by a factor of R. Our results for the no-contention case (Figure 3) properly reproduce this result. In the face of flow table contention, however, the increased churn rate from the Clogged Drain effect drives flow completion time to be greater than expected (c.f. Figure 4).

5.2 Straggling Herd and false negatives

Consider a multi-tier Web service of load balancers that handle requests, followed by processing and database tiers, as depicted in Figure 5. A load balancer receives concurrent requests with several processing tier destinations, which in turn trigger database requests. Finally, the network uses Openflow switches with limited flow table size. We expect bursts of requests to the processing tier to trigger bursts of requests to the database tier, possibly causing incast-style [10] collapse due to flow table churn depending on controller setup, topology and forwarding policy. However, the Straggling Herd effect will instead spread flow completion times over a much larger window in a scaled emulation. Consequently, requests to the database tier will be far less bursty, falsely averting the incast-style collapse that would occur in deployment.

To demonstrate this, we run a scaled-down topology consisting of three hosts: $C1$, $C2$ and $C3$. These represent a load balancer, processor and database. $C1$ and $C2$ are linked by switch $S1$. $C2$ and $C3$ are linked by switch $S2$. Links to $S2$ are fixed at 1 Gbps, while links to $S1$ are varied from 1 Mbps to 1 Gbps using `tc`. All limits are accurately enforced by `tc`; in contrast to the previous experiment, this emulation is not CPU bound. We start 1,000 flows simultaneously from $C1$ to $C2$. When a flow terminates at $C2$, $C2$ immediately starts a flow to $C3$.

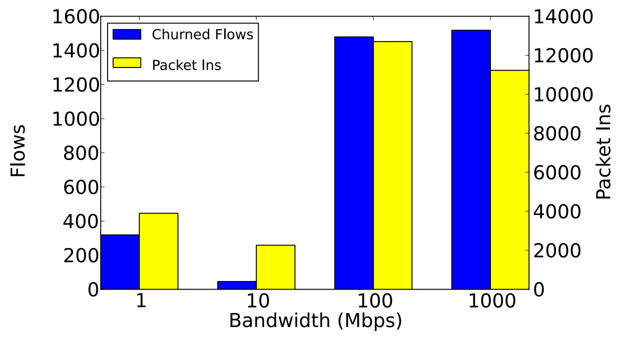

Figure 6: Packet ins from $S2$ vs. $S1$ link speed.

While we only use three hosts and two switches in this example, it represents a general class of experiments with arbitrary numbers of machines at each tier. The salient aspect of our configuration is the bottlenecked links from load balancer to processor, and bottlenecked flow tables.

We measure packet-in events at the controller from $S2$, and the resulting number of flows displaying flow table churn. Figure 6 shows that practical link speeds at $S1$ (e.g., 100 Mbps and 1 Gbps) result in significant flow table churn at $S2$, with bursts of requests from $C1$ to $C2$ causing bursts from $C2$ to $C3$ leading to large numbers of packet-in events at the controller. As emulated link speed drops, though, flow terminations from $C1$ to $C2$ become less bursty. Requests from $C2$ to $C3$ become less bursty as well, and the incast effects disappear. Hence, the scaled emulation fails to predict the significant flow-table churn and resulting decreased performance that a real deployment is likely to experience.

6 Discussion

One obvious solution to these issues is to avoid entirely the need to scale by increasing emulation scale. We are exploring ways to extend Modelnet to provide higher capacity by means of a distributed multi-machine emulator. However, such an approach remains limited by hardware cost; if we had enough capacity to emulate a large network at full bandwidth, then we will have largely negated the cost benefit of emulation. Thus, the problem of handling these artifacts remains important.

While these artifacts impact buffer contention, scaling buffer sizes is not a fix. One artifact induces unrealistic churn, while the other prevents it. Tweaking buffers to mitigate one worsens the other. Scaling experiments by reducing flow size or quantity risks changing dynamics further and providing unhelpful results.

We must deal with the root cause: by changing link bandwidths, we effectively dilate time in the network, without scaling hosts that use the network. For the clogged drain, one might argue for scheduling flows so that flow arrival rate is dilated as well. However, this implicitly induces dependencies between hosts that do not exist in real life. Time dilation has been used to trick an end host into thinking it is running on a faster network than it is actually connected to. What we need instead is a mechanism that effectively slows down end-host actions, so it does not outpace the network.

The Straggling Herd problem occurs because packets that are processed simultaneously in real life are handled in distinct batches in emulation. Batching packets and delivering them simultaneously to end hosts would be a non-solution, since it would discard per packet latency correctness and increase the period for any given emulated link where it is not getting serviced, by the same factor as the batch size. Increasing this period would reduce simultaneity between different emulated links, worsening the problem we wish to solve. Reducing effective end-host CPU rate to correspond with network slowdown might also remove this artifact.

Not slowing down end host CPUs yields another source of confusion. Put simply, there is nothing to distinguish whether we are emulating a slow network at full speed, or a fast network with scaled down links. The clogged drain problem manifesting can either be considered realistic with a slow network, or a false positive prediction if we were emulating a fast network.

Unfortunately, slowing down end-host CPU rate is not straightforward. Clustering virtual end hosts on a single host provides each with bursty access to a full speed CPU, instead of continuous access to a slower CPU. Shrinking scheduling quantums could help, but would cause significant context switching overhead.

7 Conclusion

The ability to emulate large-scale SDN networks would provide a way to reduce the risk in deploying large scale datacenters with complicated network control schemes. Past experience with network and distributed system emulation suggests that good results could be derived via shrinking a network for a variety of metrics and scenarios. However, this work shows that shrinking an emulated software defined network beyond certain limits can trigger errors that can compromise the validity of any analysis performed and any conclusions derived from the emulation.

Acknowledgements

This work is supported in part by the National Science Foundation through grant CNS-1018808.

References

[1] Openflow: NEC switch. `http://www.openflow.org/wp/switch-nec/`.

[2] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proc. NSDI* (2010), USENIX.

[3] Gupta, D., Vishwanath, K. V., McNett, M., Vahdat, A., Yocum, K., Snoeren, A. C., and Voelker, G. M. DieCast: Testing distributed systems with an accurate scale model. *ACM Transactions on Computer Systems 29*, 2 (May 2011), 4:1–4:48.

[4] Gupta, D., Yocum, K., McNett, M., Snoeren, A., Vahdat, A., and Voelker, G. To infinity and beyond: Time warped network emulation. In *Proc. SOSP* (2005), ACM.

[5] Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., and McKeown, N. Reproducible network experiments using container-based emulation. In *Proc. CoNEXT* (2012), ACM, pp. 253–264.

[6] Heller, B., Sherwood, R., and McKeown, N. The controller placement problem. In *Proc. HotSDN* (2012), ACM, pp. 7–12.

[7] Levin, D., Wundsam, A., Heller, B., Handigol, N., and Feldmann, A. Logically centralized? state distribution trade-offs in software defined networks. In *Proc. HotSDN* (2012), ACM, pp. 1–6.

[8] McGeer, R. A safe, efficient update protocol for openflow networks. In *Proc. HotSDN* (2012), ACM, pp. 61–66.

[9] Pan, R., Psounis, K., Prabhakar, B., and Wischik, D. Shrink: A method for enabling scaleable performance prediction and efficient network simulation. *IEEE/ACM Transactions on Networking 13*, 5 (2005), 975–989.

[10] Phanishayee, A., Krevat, E., Vasudevan, V., Andersen, D., Ganger, G., Gibson, G., and Seshan, S. Measurement and analysis of tcp throughput collapse in cluster-based storage systems. In *Proc. FAST* (2008), USENIX.

[11] Vahdat, A., Yocum, K., Walsh, K. K., Mahadevan, P., Kostic, D., Chase, J., and Becker, D. Scalability and accuracy in a large-scale network emulator. In *Proc. OSDI* (2002), USENIX, pp. 271–284.