An SDN-inspired Model for Faster Network Experimentation

Eder L. Fernandes
Queen Mary, University of London
e.leao@qmul.ac.uk

Gianni Antichi
University of Cambridge
gianni.antichi@cl.cam.ac.uk

Ignacio Castro
Queen Mary, University of London
i.castro@qmul.ac.uk

Steve Uhlig
Queen Mary, University of London
steve.uhlig@qmul.ac.uk

ABSTRACT

Assessing the impact of changes in a production network (e.g., new routing protocols or topologies) requires simulation or emulation tools capable of providing results as close as possible to those from a real-world experiment. Large traffic loads and complex control-data plane interactions constitute significant challenges to these tools. To meet these challenges we propose a model for the fast and convenient evaluation of SDN as well as legacy networks. Our approach emulates the network's control plane and simulates the data plane, to achieve high fidelity necessary for control plane behavior, while being capable of handling large traffic loads. We design and implement a proof of concept from the proposed model. The initial results of the prototype, compared to a state-of-the-art solution, shows it can increase the speed of network experiments by nearly 95% in the largest tested network scenario.

CCS CONCEPTS

• **Networks** → **Network simulations**; *Programmable networks*; • **Computing methodologies** → *Discrete-event simulation*;

KEYWORDS

Discrete Event Simulation, Network Emulation, Software Defined Networking

1 INTRODUCTION

Computer networks have become incredibly and unexpectedly large infrastructures, frequently underpinning a wide range of critical activities. While experimentation and innovation in these infrastructures is rather desirable, innovation (e.g., new routing protocols) or even simple alterations (e.g., changes in the topology or its configuration) result in often hard to predict behaviors. Experimenting on a real production network is problematic, and the sheer size of some of these networks can be overwhelming.

While physical testbeds are one obvious solution, they can be too costly, complex, and may suffer from tighter space constraints than the typically geographically distributed network under study. This makes the use of testbeds limited to very small-scale scenarios.

Network simulators and emulators have been extensively adopted as simpler and more accessible option for complex topologies. However, choosing the best approach between simulation or emulation is far from easy. Indeed, they both come with their own benefits and drawbacks. **Simulators** rely on mathematical and discrete-event models of reality, to reproduce the behavior of the network. Typically, network simulators achieve high reproducibility thanks to the underlying mathematical models, e.g., finite state machines. In contrast, the intricacy of configuration is a challenge to obtain solid results, given the large state-space they aim to reproduce.

On the other hand, **emulators** provide real network stacks and thus can be used to test for real behaviors. Although they can provide more realistic results than simulations, they are resource-consuming (as much as the emulated network), effectively restricting the emulated network size, especially when run on a single machine. While distributed versions of emulators increase scalability, it increases the cost of the deployment and its complexity, by for instance, requiring coordination among multiple machines.

Table 1 shows the main available options for simulating or emulating both legacy and Software Defined Networks (SDN) [6]. *Discrete Event Simulation (DES)* tools such as NS2 [4] and NS3 [9] are a good solution to foster reproducibility of network experiments. Unfortunately, the complexity to create experiments and the time required to run large scale simulations limits its potential. Fs-sdn [2] improves simulation speed, but its scope is limited to SDN environments. In contrast, emulators like Mininet [3] bring flexibility in terms of network creation, as well as for the real applications that can be used. Sadly, resource constraints make large scale experiments practically infeasible. Mininet-VT [11] and Selena [8] improve the emulation accuracy with virtual time scheduling approaches. Unfortunately, they slow down execution and require changes the Operating System kernel. Finally, S3Fnet [5] proposes an interesting mix of *Parallel DES* and emulation, to support larger scale experiments. However it also requires kernel changes and its speed suffers under high data plane loads.

This paper proposes a combination of DES and emulation but with a completely different spirit to S3Fnet. While the latter emulates the TCP/IP networking stack for any traffic, we opted for a solution that emulates only the control plane of the network, i.e., routing and SDN control protocols. Its rationale comes from two main insights related to control plane traffic: (1) it is the only one that can alter network behavior, hence to guarantee fidelity in an experiment it is important be close to its real counterpart, (2) there are way fewer control plane messages compared to dataplane

SIGSIM-PADS'18, May 23–25, 2018, Rome, Italy

ACM ISBN 978-1-4503-5092-1/18/05…$15.00
https://doi.org/10.1145/3200921.3200942

Table 1: Advantages and drawbacks of Network Emulators and DES

Tool	Type	Advantages	Drawbacks
NS2/NS3	DES	High reproducibility	Slower as scale increases, complex configuration
Mininet	Emulator	Flexible, real network stacks	Scale limited by machine resources
Mininet-VT/ Selena	Virtual Time Emulator	Increases emulation accuracy	Changes to kernel/hypervisor, slower execution
fs-sdn	DES	Fast and lightweight	SDN only, controller coupled to simulation
S3Fnet	PDES/Emulator	Large scale	Changes to kernel, slows down under high load

traffic, so approximating/aggregating the dataplane is sufficient. Our solution is inspired by the SDN approach of decoupling control from data plane. In a nutshell, we propose a novel approach that emulates the network control plane and simulates the data plane. Our design speeds up simulation while keeping control plane fidelity. Additionally, our approach can be used to perform experiments on both legacy and SDN networks. We present a prototype and show how it succeeds in achieving rather low execution times that barely grow in a test with increasing network sizes.

2 A MODEL FOR CO-EXISTENCE OF SIMULATION AND EMULATION

Inspired by the original goal of SDN, our approach decouples control and dataplane, by emulating the former and simulating the latter. Such a separation introduces a key challenge: designing a solution that allows the simulation and emulation layers to co-exist. In our solution, both abstractions use different traffic granularities and concept of time. Whereas the former uses coarse granularity abstractions, e.g., flow models, and adopt a pure DES approach, the latter works in a per-packet manner and relies on a *Fixed Time Increment (FTI)* mechanic, to reproduce a continuous time environment. The platform thus runs DES when there is no interaction with the emulated control plane, and the FTI mode when control messages events are executed. The resulting model is scalable, as the amount of control traffic, i.e., routing protocol packets, SDN control messages, in networks is much lower than the data traffic. The two modes operate as follows:

Pure DES. In traditional DES, the simulation clock advances to the time of the most recent event. Algorithm 1 shows that the events are executed in DES mode until an event from the control plane happens. After such event, the simulator switches to the FTI mode to execute in real-time. The event with the highest timestamp possible in the system (e.g., in a 64 bits system the value is an unsigned 64 bits integer) signals the end of the simulation.

Fixed Time Increment. When triggered, the clock advances in equal time intervals, and all the events scheduled before the current time are executed (Algorithm 2). A user-defined parameter (**controller idle interval**) defines the interval required to switch back to the DES approach, when the interaction between the simulated and emulated planes is considered done. It is important to define an appropriate interval due to possible inconsistencies in the simulation. If the controller idle timeout is lower than the time required for the control plane to react to a control plane event, the simulation will switch to DES and execute the next event that might be ahead of a control plane reply. On the other hand, a too large value will slow down the execution, as the duration of the FTI mode will last longer than necessary.

Algorithm 1 Discrete Event Simulation mode

```
1:  while not fixed_mode do
2:      event ← scheduler_retrieve(scheduler)
3:      sim_clock ← event.time
4:      handle_event(event)
5:      if event.type = CTRL_MSG then
6:          fixed_mode ← true
7:          Switch to FTI mode
8:      else
9:          if event.type = END then
10:             Finish the Simulation
11:         end if
12:     end if
13: end while
```

Algorithm 2 Fixed Time Increment mode

```
1:  while fixed_mode do
2:      sim_clock ← sim_clock + 100
3:      event ← scheduler_retrieve(scheduler)
4:      while event.type ≠ END and event.time ≤ sim_clock do
5:          handle_event(event)
6:          if event.type = CTRL_MSG then
7:              last_ctrl ← event.time
8:          end if
9:          event ← scheduler_retrieve(scheduler)
10:     end while
11:     if (sim_time − last_ctrl) ≥ idle_interval then
12:         fixed_mode ← false
13:         Switch to DES mode
14:     end if
15: end while
```

2.1 Dataplane Traffic Model

To be able to experiment with large topologies, we use a flow-level representation of network traffic, consisting of packets aggregated by common headers, but potentially differing in size (similarly to fs-sdn [2]). To account for traffic loss, we split flows' arrival and departure into two different events. Congestion occurs when the aggregated size of concurrent flows and same (output-port) destination arrive simultaneously to a node with insufficient output-port capacity. Currently, the system cannot compute the packet

loss referred to a specific flow, so it randomly distributes the excess of traffic that will be dropped among all the competing flows.

3 A DESIGN FOR PAST AND FUTURE CONTROL PLANES

In this section we present the design of a simulator that uses the proposed model. Figure 1 pictures the general architecture of the solution. The top part represents the emulated network control plane. It supports future proof approaches such as the those based on a logically centralized SDN control or legacy protocols such as the *Border Gateway Protocol (BGP)*. The Connection Manager (CM) sits in-between the control and data plane, and delegates messages generated by events from the simulation to the emulation and vice versa. To avoid possible loss of performance caused by frequent changes from DES to FTI, some control plane traffic, like the connection of an SDN controller with switches and common *keep alive* messages are handled by the CM and never reach the simulation side. The bottom part of the figure depicts the simulated data plane. The Scheduler is responsible for adding and retrieving events from the Event Queue in the correct order. At the Event Handler, five events are processed: start of applications, arrival and departure of flows, control messages from and to the control plane. The last two trigger the mode change from DES to FTI. The Topology block contains the simulated logic of network nodes such as hosts, routers and SDN switches. Statistics from the simulation are stored in a file or an independent database, so it can be accessed by the control plane.

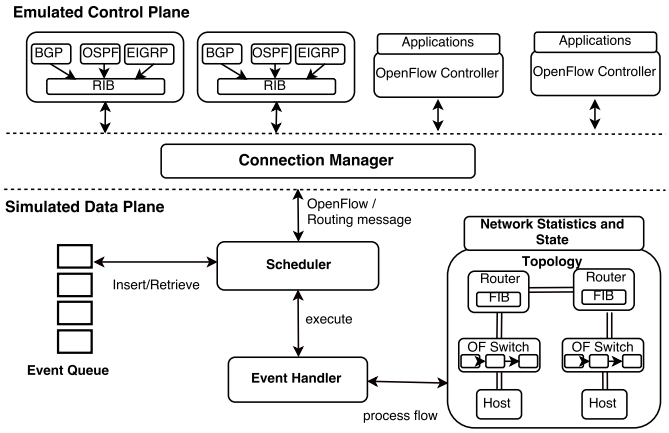

Figure 1: General Architecture of the Simulator

In the following, we show how our approach can work with both SDN solutions or legacy control plane protocols.

SDN. This is the most natural candidate, as our model is inspired in the control-data plane decoupling. In our design, the switches are nodes implementing OpenFlow [7], the most widespread SDN realization. Controllers are part of the control plane, thus can be real and independent software instances. Consequently, when interactions between SDN controllers and the network are needed, the operation mode changes from DES to FTI. This change enables the controller to interact with the simulator in near real-time. Furthermore, new events from the controller are inserted in the event queue, ahead of data plane events with the same timestamp. This ensures the timely occurrence of control plane events preventing data plane disruptions (e.g., traffic loss). We now illustrate this with the example of a controller reacting to an event from the data plane:

- The simulator starts in the DES mode and executes an event where a host sends a flow to a switch connected to it.
- The switch does not have an OpenFlow rule to handle the flow. Following the default no-match action, a message must be sent to the controller.
- When the message is executed, the system changes to FTI operation. The controller handles the message and sends a reply message which is added to the event queue.
- The reply from the controller is then executed, and the controller does not interact with the platform until the controller's idle timeout. The system switches then back to DES.

BGP. Applying a legacy control plane to our architecture can be done by enabling as many controllers as the data plane routers in the topology. Indeed, the logical difference between a logically centralized solution such as SDN and a distributed one, such as BGP, is that the former presents a mapping of one or more controllers for one to many switches, while in the latter case, each router has its own embedded controller. In this scenario, the emulated part runs lightweight BGP speakers (e.g., ExaBGP [1]) that allow real BGP sessions in the control plane to generate the *Routing Information Base (RIB)*. From the RIB, the *Forwarding Information Base (FIB)* can be derived and installed in the data plane.

In the BGP design, announcements are sent to the control plane from the simulation side, triggering the FTI mode. The CM will instruct the emulated BGP speaker to create a real announcement to its destination. The receiving BGP peer will then store the reachability information and build the local RIB. Finally, a FIB can be derived and installed in the respective simulated router as an event coming from the control plane. Upon BGP convergence, the system is expected to resume to the pure DES mode.

4 PROOF OF CONCEPT AND EVALUATION

In this section we present and evaluate a prototype of our approach, which is publicly available [2]. The prototype is implemented in C with Python bindings for the *Application Programming Interface (API)*, to enable simpler creation and configuration of network scenarios. We abstract the connection of real controllers to the simulated data plane using an OpenFlow driver [10] in the CM. The CM runs in an independent thread with access to the Scheduler, so the messages received from the controller can be added to the Event Queue. Each operation mode (i.e., DES and FTI) has its own thread that implements Algorithms 1 and 2, respectively. The thread of the FTI runs periodically every 100 milliseconds, although the time between two executions may vary slightly due to the system scheduling. For this reason, time increases in FTI as the difference between the system clock and the last execution of the thread.

We compared our solution with Mininet, as it is the de-facto tool for SDN experimentation. Specifically, we assessed our solution in

[1] https://github.com/Exa-Networks/exabgp
[2] Repository: https://github.com/ederlf/paper-sigsim-pads-18

terms of speed and reproducibility[3]. Our experiment runs a traffic pattern five times in a k-ary Fat-Tree: a common topology for data centers. Each host sends 1Mbps of UDP traffic to a single destination over a 60 seconds period. The expected aggregated bandwidth per second, for the whole experiment, is equal to the number of hosts. The topology is rather appropriate for large-scale networks: it is composed of three layers of switches (edge, aggregation and core) connecting hosts in a number of pods equal to a variable k. The number of hosts increases exponentially with k, while the number of switches nearly doubles. The network consists of SDN switches controlled by one SDN controller that runs *Equal-Cost Multipath (ECMP)* to load-balance traffic among the links. All the links have enough bandwidth to avoid congestion, so the total aggregate traffic can be observed. To demonstrate that our solution runs fine in modest hardware, the experiments are performed in a computer with two physical cores at 2.9 GHz of speed and 8GB of RAM.

Our solution clearly achieves its goal of low execution times. Table 2 shows how our the execution times (in seconds) are almost constant as the network grows. Moreover, it takes just about 5% of the time it takes for Mininet to reproduce the largest scenario ($k = 8$).

Table 2: Comparison of the total execution time for 5 trials of experiments in a Fat Tree topology controlled by ECMP implemented as an SDN application.

k/switches/hosts	Our Approach	Mininet
4/20/16	57s	416s
6/44/54	58s	562s
8/80/128	60s	1141s

For reproducibility, we compare the average aggregated bandwidth in the 5 trials. The expected value for a test where k equals 6 and 8 is 54Mbps and 128Mbps, respectively. Figure 2 shows that our implementation reproduces the expected amount of traffic in both cases. On the other hand, Mininet struggles as the topology grows, because it requires much more computing resources, showing less traffic than expected in half of the experiment when $k = 8$.

Note that we also checked other tools from table 1. We experimented with an OpenFlow module for NS3 [1] but did not execute five trials per network size because a single run with with $k = 8$ took more than 50 minutes to finish. As for the other tools, the configuration overhead to implement the tests made them too time-consuming to obtain results in time for this paper. We hope to perform a thorough comparison in our future work.

5 CONCLUSION

In this paper, we presented a platform for fast and accurate reproduction of network experiments. Our architecture, inspired by the SDN approach of decoupling control and data planes, is based on an emulation layer for the control plane traffic, and a simulation layer for the dataplane traffic. We discussed how the proposed solution can be used to perform experiments on both legacy and SDN-enabled networks. We implemented a proof of concept and assessed its potential by comparing speed and reproducibility capabilities, against state-of-the-art solutions such as Mininet. We found that our platform provides control plane fidelity, while delivering results nearly 95% faster than the most commonly used solution.

[3]This aspect is even more important than speed. With this test we look into the ability of a tool to reproduce results given same inputs.

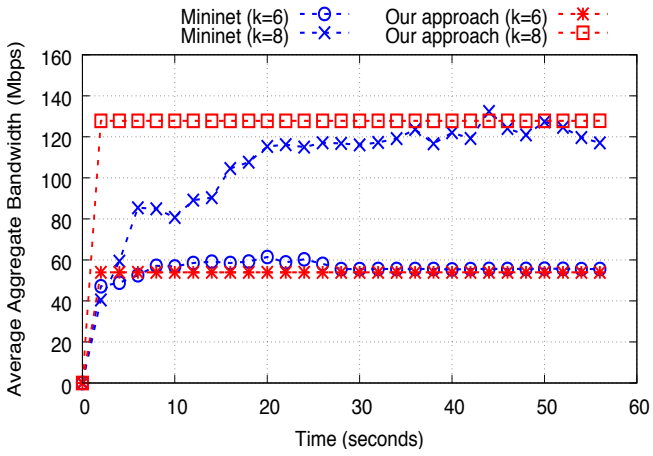

Figure 2: Comparison of the average aggregated bandwidth after 5 test trials using our approach and Mininet. The expected average value, during the whole experiment, for k=6 is 54Mbps and for k=8 is 128Mbps.

ACKNOWLEDGMENTS

This research is supported by the UK's Engineering and Physical Sciences Research Council (EPSRC) under the EARL: sdn EnAbled MeasuRement for alL project (Project Reference EP/P025374/1).

REFERENCES

[1] Luciano Jerez Chaves, Islene Calciolari Garcia, and Edmundo Roberto Mauro Madeira. 2016. OFSwitch13: Enhancing Ns-3 with OpenFlow 1.3 Support. In *Proceedings of the Workshop on Ns-3*. ACM, New York, NY, USA.
[2] Mukta Gupta, Joel Sommers, and Paul Barford. 2013. Fast, Accurate Simulation for SDN Prototyping. In *Hot Topics in Software Defined Networking (HotSDN)*. ACM.
[3] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. 2012. Reproducible Network Experiments Using Container-based Emulation. In *Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. ACM.
[4] Teerawat Issariyakul and Ekram Hossain. 2010. Introduction to Network Simulator NS2. Springer Publishing Company, Incorporated.
[5] Dong Jin and David M. Nicol. 2013. Parallel Simulation of Software Defined Networks. In *Principles of Advanced Discrete Simulation (PADS)*. ACM.
[6] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. 2015. Software-Defined Networking: A Comprehensive Survey. (2015).
[7] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. In *Computer Communication Review, Volume: 38, Number: 2*. ACM.
[8] Dimosthenis Pediaditakis, Charalampos Rotsos, and Andrew William Moore. 2014. Faithful Reproduction of Network Experiments. In *Architectures for Networking and Communications Systems (ANCS)*. ACM.
[9] George F. Riley and Thomas R Henderson. 2010. The ns-3 Network Simulator. In *Modeling and Tools for Network Simulation*. Springer Berlin Heidelberg.
[10] Allan Vidal, Christian Esteve Rothenberg, and Fábio Luciano Verdi. 2014. The libfluid OpenFlow Driver Implementation. In *32nd Brazilian Symposium on Computer Networks (SBRC)*. SBC, 8.
[11] Jiaqi Yan and Dong Jin. 2015. VT-Mininet: Virtual-time-enabled Mininet for Scalable and Accurate Software-Define Network Emulation. In *Symposium on Software Defined Networking Research (SOSR)*. ACM.