OPSDN: an enhanced SDN simulation framework for OPNET Modeler

Zequn Jia
(zachary@bjtu.edu.cn)

Yantao Sun
(ytsun@bjtu.edu.cn)

Qiang Liu
(liuq@bjtu.edu.cn)

Abstract

Traditional SDN simulators/emulators cannot provide acceptable simulation capabilities in large-scale or high-traffic network scenarios. We propose OPSDN, a simulation framework that unifies the event scheduling of the existing Ryu controller framework with the traditional OPNET DES (Discrete Event Simulation) engine. In OPSDN, the SDN controller events are also triggered by the DES engine, ensuring the consistency of the simulation progress between the two sides, thus obtaining more accurate simulation results. Additionally, OPSDN provides Ryu-like APIs and can reuse part of the Ryu libraries. Our simulation results show that the OPSDN controller can achieve the same control effect as the typical Ryu controller, and avoid unexpected TCP retransmissions due to the inconsistent simulation speed. We hope OPSDN will help researchers achieve more accessible large-scale SDN network simulations.

Keywords

Large-scale SDN simulation, Ryu, OPNET

1. Motivation and significance

Real SDN controllers are adopted in most of the common SDN simulators and emulators (e.g., OPNET, Mininet[1], Estinet[2], etc.). These network simulators and emulators provide mechanisms, like SITL(System-in-the-loop) [3] in OPNET, to connect an actual SDN controller software to the simulated network, which allows the interaction between the simulated OpenFlow switches and the SDN controllers. This kind of solution ensures that the implementation of the controller in the simulation is close to the implementation in the production environment, making the simulation highly accurate and trustworthy. These simulation and emulation tools enable the simulation of an SDN network on a laptop-level device, which is of great significance for learning and researching SDN networks.

However, with further research on the SDN networks, especially the application of SDN in large-scale networks such as data-center networks, the scale of the network to be simulated in the SDN research proliferates. Many questions emerge when conducting large-scale and high-traffic network simulations using the network simulation tools as mentioned above.

The best-known SDN research tool, Mininet, relies on the performance of the host workstation for simulation. When emulating large-scale networks with high link bandwidths and high traffic, the computational complexity exceeds the capacity of the simulation workstation. Problems such as the inability to create network interfaces and slow response may occur. Distributed simulation tools like

MaxiNet[4] have been proposed to solve this problem. Nevertheless, distributed simulation tools have higher requirements in both hardware and building environment. Another problem of distributed simulation that cannot be ignored is the distortion of performance results due to the interconnection bound among simulation nodes.

As mentioned in [5], discrete-event simulation(DES) mechanism-based simulation tools like OPNET have multiple advantages in SDN simulation, e.g., better delay and jitter reflection. These tools have the ability to simulate large-scale networks in non-SDN network scenarios. Benefits from the DES engine, these tools can simulate higher computational complexity networks via the time dilation technique[6]. However, in the officially recommended SDN scenario shown in Figure 1, the simulation software based on the DES mechanism suffers from several problems.

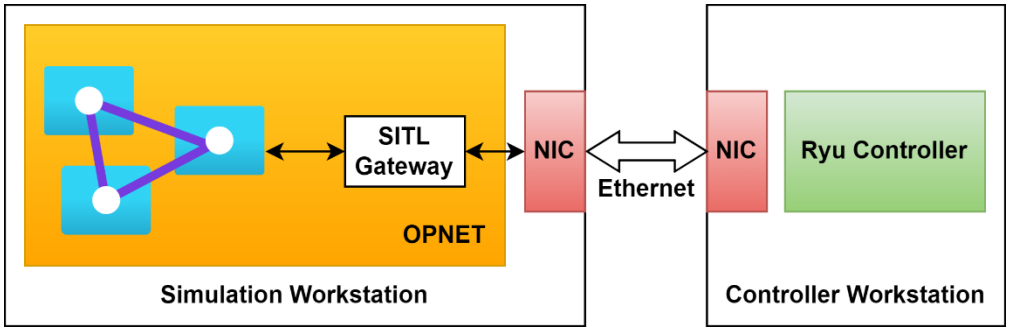

Figure 1 OPNET recommended SDN simulation architecture.

- The switches within OPNET and the real controllers run under separate event mechanisms. Event scheduling for the simulated switches in OPNET is OPNET's DES engine, while controllers such as Ryu[7] rely on its internal event mechanism (e.g., Eventlet[8]). The event scheduling mechanisms are not consistent between the two components. Currently, the synchronization between the two sides requires that OPNET run in real-time mode.
- OPNET's SITL gateway connects the simulated switches to the real controller. However, OPNET has a simulation speed upper bound due to the performance limitations of the simulation workstation, especially in scenarios with large number of nodes and traffic. When the simulation complexity reaches this limitation, the OPNET simulation will run behind real time, while external controllers are unable to sense and adapt to this speed change. The speed mismatch between the two sides will impact the simulation behaviours, causing unexpected distortion of the simulation results.
- There is a throughput bottleneck in OPNET's SITL gateway [9], as it utilizes the Libpcap[10] library for packet capturing and sending. This performance bottleneck may limit the interaction between switches and controllers. The benchmark tests[9,11] show that the transmission capability of the SITL interface is no more than 100Mbps, which is much smaller than the actual bandwidth of the controller in some scenarios like data-center networks.

In addition, the SITL interface also suffers from other minor glitches like instability. Due to the problems above, large-scale network simulation, especially in a single workstation, becomes problematic in simulation tools like OPNET and Mininet. The lack of such solutions hinders the research of various algorithms in scenarios such as large-scale networks and performance-sensitive networks.

We propose an innovative solution in OPNET to this problem. In this solution, we unify the event scheduling mechanism of controllers with simulated switches'. The unified event scheduling mechanism will allow an as-soon-as-possible simulation progress mode, which allows the simulator to slow down the simulation speed but guarantee the simulation accuracy in high-complexity simulations. In contrast to implementing a controller model from scratch inside OPNET, we port a common OpenFlow controller, Ryu, to the DES engine, i.e., it will provide APIs similar to Ryu's, ensuring that most Ryu applications can run in our framework with simple modifications.

This compatibility allows 1) all kinds of existing Ryu Apps could be used upon our framework, 2) controller applications after simulation can be simply ported back to Ryu for testing and deployment, 3) various libraries provided by Ryu (e.g., OpenFlow support) can be reused in our framework without reinventing the wheel, and 4) controller development can still be done in the simple Python language with the support of Python's substantial open-source packages instead of using C/C++ in OPNET for development.

2. Software Description

The event framework in the Ryu framework was re-implemented to unify the event mechanism of OPNET and Ryu, abandoning the original Eventlet-based event mechanism. In this way, all events in the controller are also triggered by OPNET. Currently, we focus on two types of events in OPNET: packet interrupts and self-interrupts. Packet interrupt events are triggered by the arrival of packets. When a packet arrives at the controller, the corresponding event handler will be called and the event will be dispatched according to the type of the packet, and various user-defined event handlers will be triggered. Self-interrupt events are used to implement the OpTimer mechanism. Python built-in timer should be avoided as there might be a difference between simulation time and real time. Therefore, it is essential to establish a timer mechanism based on OPNET simulation time. OpTimer is provided as an API for creating events executed at the user-specified simulation time.

2.1. Software architecture*:*

The OPSDN software consists of the OPNET controller model, a bridging dynamic link library and a Python package OpSdnPy. The overall software architecture is shown in Figure 2.

The OPNET controller model includes Controller Node Model, Controller Process Model and Controller Support External File. The Controller Node Model is a node model modified from the OPNET Ethernet workstation node model, adding a Controller Module connecting to the TCP module via the packet stream. The Controller Module can be used as an application layer module to establish a secure channel and communicate with simulated OpenFlow switches using the standard TCP/IP protocol stack in OPNET. Controller Process Model and Controller Support External File are where the OPSDN's internal logic is implemented inside OPNET. The process model is responsible for listening to TCP connections from OpenFlow switches and converting simulated packets to real packets and backwards. The converted real packets will be sent to the OPSDN-DLL Bridge and finally reach the Python SDN controller.

The OPSDN-DLL Bridge module is a dynamic link library for the OPNET. This dynamic link library creates a Python VM environment to execute the controller's Python scripts. The reason for needing a dynamic link library instead of creating a Python environment directly in the OPNET model is that we

found that in the version of OPNET we use (18.6), OPNET has an embedded Python2 environment to support some features. Python2 is too old and no longer supported, and Ryu does not run in Python2, so we prefer to use the Python3 environment. However, However, it is challenging to integrate both Python2 and Python3 in the same executable, so we use a dynamic link library to isolate the two Python environments. Thus the bridge acts as a proxy program to forward event calls from OPNET to the Python environment and backwards.

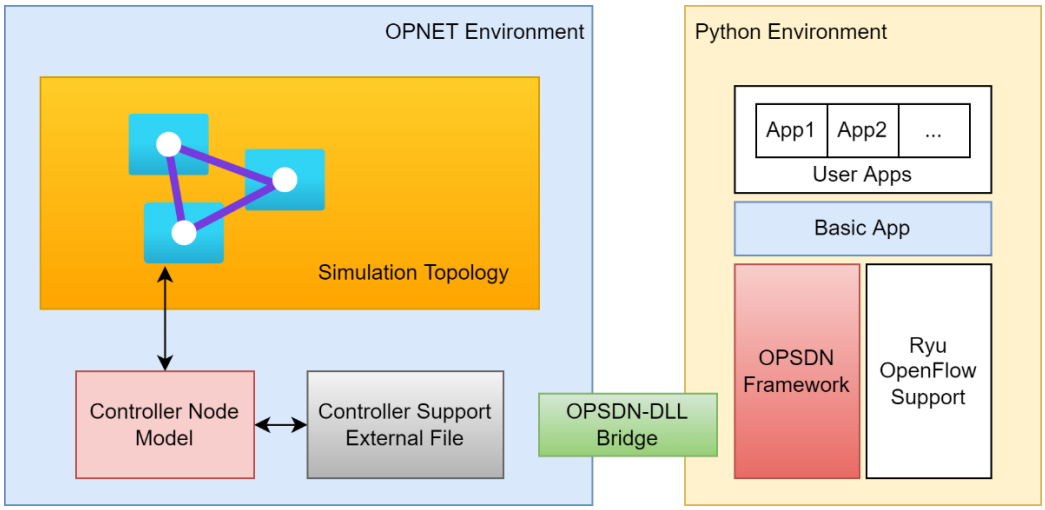

Figure 2 OPSDN architecture.

OpSdnPy is a Python package for OPSDN that we created. This package provides Ryu-like controller APIs, including Controller, Datapath, OpTimer, OpsdnApp, etc. We also implement a basic app for establishing secure connections with OpenFlow controllers. Users can create their own Controller instances and Applications based on the functionality provided by the framework. In the configuration of the OPSDN Controller Node model, users are allowed to configure user-written script files and controller classes, which will be called during the simulation.

2.2. Software functionalities:

Our software provides the following basic functionalities:

1) A complete SDN runtime environment for OPNET, without the limitations of SITL performance and network scale and traffic, which relies on OPNET's DES mechanism to provide accurate simulation results without the distortion caused by differences in switch and controller event mechanisms.
2) A configurable OpenFlow controller model. Users can directly add this controller model to the network simulation scenario by connecting it to OpenFlow switches. The model provides several configurations for the Python environment, script file locations, entry file, and entry controller class. Switching between several different controller scripts can be easily supported.
3) Python APIs for implementing the SDN controller. This framework provides three types of Python APIs to support the controller implementation. The first is the controller application API, which contains two Python decorators, *@controller_app* and *@set_ev_handler*, to indicate the controller application and event handler functions. The second is the Datapath API, which implements the common API of the class of the same name in Ryu to interact with OpenFlow switches. The third

one is OpTimer API, which implements a timer mechanism based on DES time, and can be used to implement various types of time-related events in the controller.

3. Illustrative examples

In this section, a Fat-tree topology simulation example in a data-center network is provided to demonstrate the simulation workflow and results in OPSDN.

First we build an SDN simulation scenario in OPNET software using the out-of-band approach based on the Fat-tree topology, as shown in Figure 3. Compared with the traditional SDN scenarios in OPNET, one of the main differences is that all the OpenFlow switches are eventually connected to the controller node through a router instead of the SITL gateway. We deploy full-mesh application demand traffic on all server nodes through the traffic deployment tool.

On this controller node, we configure essential controller configurations such as the location of the Python directory, and the controller script file location. The project file and the detailed code of the controller scripts are placed in the example folder. We developed two Controller Apps for forwarding in Fat-tree topology: topology_discover.py and fattree_fw.py. The *TopologyDiscover* app is used to discover the whole network topology, by periodically asking all switches to send LLDP packets to their neighbouring nodes to obtain the connection between nodes. *FattreeForward* application is responsible for packet forwarding in Fat-tree topology. Packets that switches cannot forward will be sent to the controller via Packet-In messages, and *FattreeForward* will calculate the shortest available path based on the discovered topology and then send the flow table to the switches.

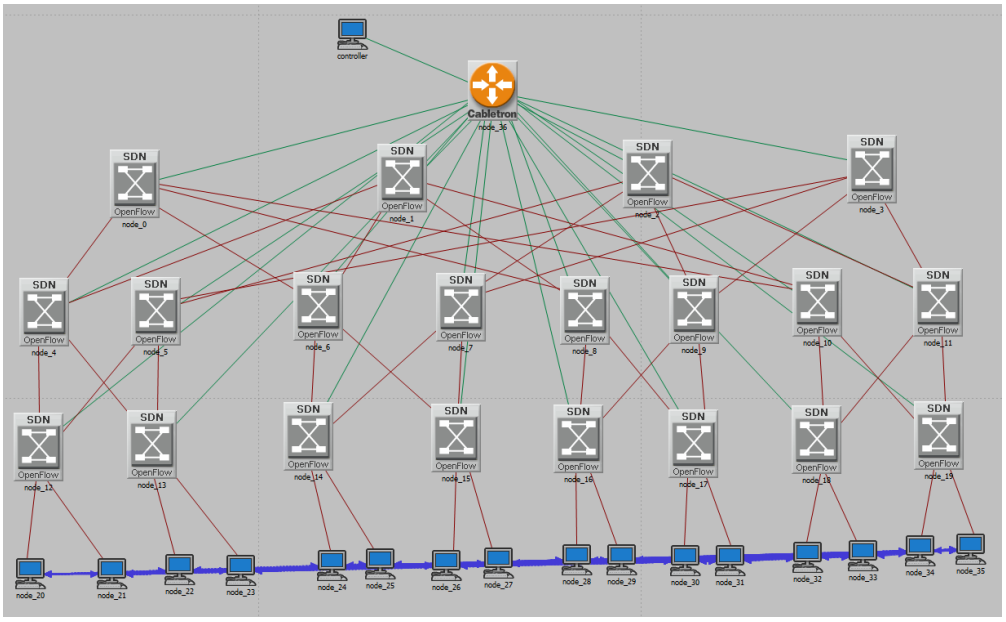

Figure 3 The Fat-tree simulation topology.

We also deploy the SITL-based SDN scenario as OPNET recommended for comparison. The simulation results are shown below.

We first test the *TopologyDiscover* app in OPSDN. The *TopologyDiscover* app will use the typical LLDP-based topology discover method to find all the switches and their links. The vis.html file will visualize the whole topology. It is demonstrated that the OPSDN controller could find all the switches, hosts and links in Figure 4, which means the essential OpenFlow message interaction works.

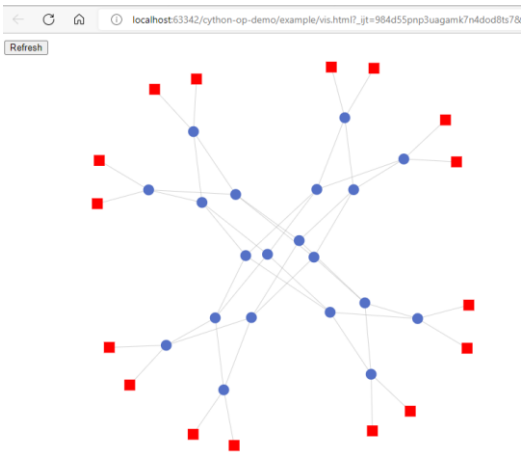

Figure 4 Topology Discover App discovers the switches, hosts and links via periodically.

Then we record the traffic of all the application demands using OPNET's DES statistics. The traffic between node_20 and node_24 is shown in Figure 5. It is demonstrated that the sent and received traffic are almost identical in both OPSDN and SITL scenarios. This means that our OPSDN controller could establish secure channel with the SDN switches, and distribute flow entries to them. Therefore our controller has the same functionality as the SITL controller and is able to support the simulation of SDN networks.

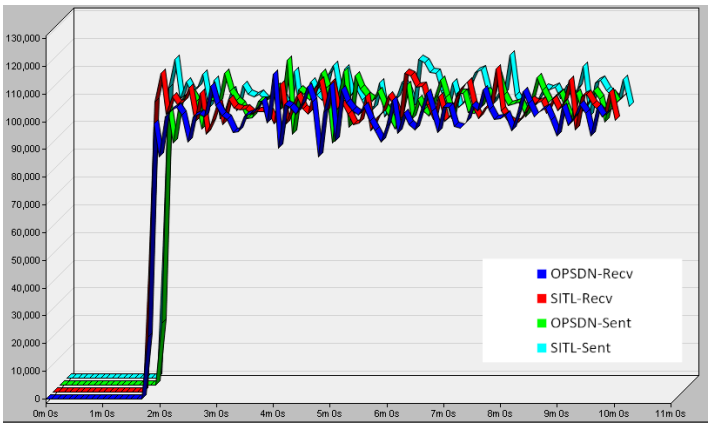

Figure 5 Traffic between node_20 and node_24 in OPSDN and SITL scenarios.

In addition, we use Wireshark to capture SITL packets during simulations, i.e., the traffic between the controller and SDN switches. When we analyse the packet trace files, TCP retransmissions are noticed when the OPNET simulation speed is slower than the real time. This is because the simulation speed is slower than the controller application and TCP Acks will be sent to the controller later than expected. Several simulations with different traffic configuration (PPS, packet per second) are conducted, and the

results are shown in Figure 6. It is demonstrated that 1) the simulation time increase as the deployed traffic grows, and 2) more TCP retransmissions are found in scenarios with higher traffic configuration. When researchers simulate performance-sensitive scenarios in OPNET, these TCP retransmissions may lead to gaps between the simulation and the real results, which in turn may affect the final conclusions. In our OPSDN scheme, both ends of the TCP connections are running in the OPNET simulator, so it is able to avoid this situation and help the researchers get simulation results closer to the real.

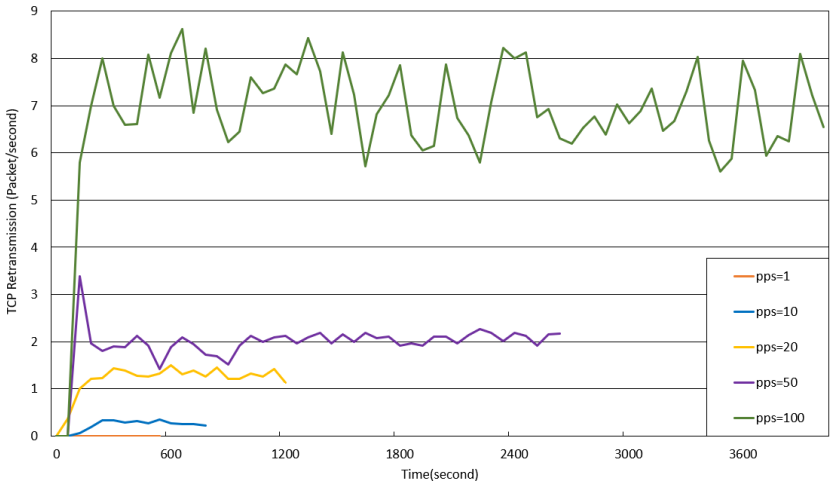

Figure 6 the TCP retransmissions rate in SITL scenarios in different traffic configurations.

4. Impact

OPSDN provides researchers with a network simulation framework that allows them to conduct large-scale SDN network simulations on laptop-level hardware and leverage the accurate simulation modelling capabilities provided by OPNET. Therefore, the main impacts of this software are as follows.

1） A hybrid simulation framework for OPNET and Ryu-like controllers is proposed to provide a unified event scheduling mechanism based on OPNET DES engine, allowing OPNET and SDN controllers to simulate simultaneously and avoid simulation distortion caused by different event mechanisms. This allows researchers to perform accurate SDN network modelling and simulation without worrying about the simulation architecture.

2） It provides a large-scale simulation mechanism running in laptop-level hardware for SDN simulation, filling the lack of large-scale network simulation capability in various types of network simulation/emulation software. This software enables an acceptable system complexity and hardware cost for simulating the large-scale network simulation. Researchers are able to perform larger-scale simulations such as large-scale SDN data-center networks and large-scale SD-MANETs. In addition, this framework can also help researchers conduct research on scheduling algorithms and topology design with large bandwidth and high traffic, which are difficult to obtain accurate results in traditional environments due to several constraints.

3） Researchers are allowed to use Python as the programming language for controllers instead of C/C++, even running in OPNET. In addition, we provide a Ryu-like controller framework that provides APIs

similar to Ryu, making it easy for controller developers to port existing Ryu Apps. Researchers could build controller algorithm prototypes rapidly.

4） The idea of combining the DES event mechanism with the event mechanism of controller frameworks such as Ryu presented in this software is general. This software verifies the feasibility of this idea, which means we can apply it to other DES-based simulation tools, such as NS3 and OMNET++, and provide more powerful tools for subsequent research and simulation of SDN.

5. Conclusions

OPSDN is an improvement for the existing SDN simulation environment, which solves the problem that OPNET cannot perform large-scale and high-traffic SDN simulation. This software builds a bridge between Ryu-like controller applications and OPNET simulation software. We hope that OPSDN can help researchers lower the requirements of large-scale and high-traffic SDN network simulation and provide ideas for other similar simulation platforms.

References

[1] B. Lantz, B. Heller, N. McKeown, A network in a laptop: Rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Association for Computing Machinery, New York, NY, USA, 2010. https://doi.org/10.1145/1868447.1868466.

[2] Shie-Yuan Wang, Chih-Liang Chou, Chun-Ming Yang, EstiNet openflow network simulator and emulator, IEEE Commun. Mag. 51 (2013) 110–117. https://doi.org/10.1109/MCOM.2013.6588659.

[3] L. Hu, J. Wang, E. Song, A. Ksentini, M.A. Hossain, M. Rawashdeh, SDN-SPS: Semi-physical simulation for software-defined networks, IEEE Sensors Journal. 16 (2016) 7355–7363. https://doi.org/10.1109/JSEN.2016.2561969.

[4] P. Wette, M. Dräxler, A. Schwabe, F. Wallaschek, M.H. Zahraee, H. Karl, MaxiNet: Distributed emulation of software-defined networks, in: 2014 IFIP Networking Conference, 2014: pp. 1–9. https://doi.org/10.1109/IFIPNetworking.2014.6857078.

[5] S. Lee, J. Ali, B. Roh, Performance Comparison of Software Defined Networking Simulators for Tactical Network: Mininet vs. OPNET, in: 2019 International Conference on Computing, Networking and Communications (ICNC), IEEE, Honolulu, HI, USA, 2019: pp. 197–202. https://doi.org/10.1109/ICCNC.2019.8685572.

[6] P.D. Hyden, Time dilation: Decreasing time to decision with discrete -event simulation, PhD Thesis, 2003. https://www.proquest.com/dissertations-theses/time-dilation-decreasing-decision-with-discrete/docview/305346223/se-2?accountid=32743.

[7] Ryu, Ryu SDN Framework. (2022). https://ryu-sdn.org/ (accessed June 19, 2022).

[8] Eventlet, Eventlet Networking Library. (2022). https://eventlet.net/ (accessed June 19, 2022).

[9] B. Van Leeuwen, D. Burton, U. Onunkwo, M. McDonald, Simulated, emulated, and physical investigative analysis (SEPIA) of networked systems, in: MILCOM 2009 - 2009 IEEE Military Communications Conference, 2009: pp. 1–7. https://doi.org/10.1109/MILCOM.2009.5380118.

[10] Libpcap, TCPDump & Libpcap. (2022). https://www.tcpdump.org/ (accessed June 19, 2022).

[11] OPNET SITL benchmark, (2022). https://github.com/ZacharyJia/OPNET-SITL-Benchmark (accessed June 20, 2022).