SDN-TO: Software-Defined Networking Traffic Optimization with OpenDaylight and Mininet

Mathieu Stenzel[1], Dalia Lablack[2], and Jaehoon (Paul) Jeong[3]
[1]Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
[2]Télécom SudParis, Évry, France
[3]Sungkyunkwan University, Suwon, Republic of Korea
Email: mathieu.stenzel@fau.de, lbk.dalia@gmail.com, pauljeong@skku.edu

***Abstract*—As networks continue to grow in scale and complexity, the dynamic and centralized control offered by Software-Defined Networking (SDN) is increasingly vital for optimizing performance and ensuring resilience. With the adoption of SDN, network traffic can be dynamically rerouted to avoid congestion and failures, while Quality of Service (QoS) can be adjusted in real time to prioritize critical applications. This paper leverages OpenDaylight (ODL) as the SDN controller and Mininet for network emulation, using the HTTP-based communication protocol RESTCONF to manage and optimize traffic flows. By developing an intelligent and adaptive SDN framework, this work contributes to the pursuit of scalable, efficient, and resilient networks in an era of unprecedented connectivity.**

***Index Terms*—Software-Defined Networking, Network Traffic Optimization, OpenDaylight, Mininet, Real-Time Telemetry.**

I. Introduction

In modern networks, the increasing demand for high-speed data transmission and reliable connectivity presents significant challenges in traffic management and resource optimization. Traditional static network configurations often struggle to adapt to dynamic changes, such as congestion and link failures, leading to poor performance and inefficient resource utilization.

Software-Defined Networking (SDN) has emerged as a transformative approach, offering centralized control and programmability of network infrastructure [1]. By decoupling the control plane from the data plane, SDN enables dynamic reconfiguration of network paths, allowing for more efficient traffic management and improved QoS. This paper leverages OpenDaylight (ODL) [2], which is a leading open-source SDN controller, to implement a network traffic engineering system capable of optimizing traffic flows in real time.

The goal of this paper is to design and develop an SDN Traffic Optimization (SDN-TO) System that dynamically manages network traffic based on performance metrics and conditions. Mininet [3] as a widely used network emulator is employed to emulate the network environment, providing flexibility in testing various topologies and configurations. The communication between the Mininet emulator and the ODL controller is facilitated through the RESTCONF protocol [4], with YANG data models defining the configuration schema [5].

The proposed SDN-TO system automates key tasks, such as routing and prioritizing traffic, improving network reliability, efficiency, and performance. By continuously monitoring network latencies and adjusting traffic paths accordingly, the system ensures optimal utilization of resources and minimal service disruption.

The remainder of this paper is organized as follows. Section II summarizes related work. Section III describes the design of our SDN traffic optimization system. Section IV describes the performance of our system. Finally, in Section V, we conclude this paper along with future work.

II. Related Work

The field of network traffic engineering has evolved significantly with the advent of SDN, which decouples the control plane from the data plane, enabling more dynamic and programmable networks. This section outlines key related work and technologies for network traffic engineering in SDN networks.

A. Software-Defined Networking and OpenDaylight

SDN emerged as a paradigm to improve network flexibility and efficiency. Early SDN work in [6] introduced the OpenFlow protocol [7], enabling centralized control of network devices. ODL, an open-source SDN controller, extends this vision by providing a modular platform that supports various southbound protocols, such as OpenFlow, NETCONF, and RESTCONF [4], [7], [8].

The NETCONF protocol [8] provides a standardized mechanism for network configuration, offering transaction-based operations for reliability. On top of this, the RESTCONF protocol [4] enables a simpler HTTP-based network management interface, facilitating integration with modern applications. Finally, YANG [5] offers a data modeling language that ensures network configurations with compatibility and consistency via NETCONF and RESTCONF.

B. Traffic Engineering and Optimization Strategies

Traffic engineering techniques focus on optimizing the flow of data through a network to enhance performance and reliability. Classic approaches, such as Multiprotocol Label Switching (MPLS) [9], [10], rely on predetermined paths. However, SDN-based approaches leverage real-time data to make dynamic routing decisions. Note that algorithms like Dijkstra's and A* have been widely used for shortest path computation [11]. They are used for network traffic engineering in the SDN networks.

III. Design

The design of our network traffic engineering system leverages a modular architecture, integrating several key components that work together to optimize and dynamically manage traffic flows in an emulated SDN environment. The source code of our SDN-TO system is available at our GitHub repository of https://github.com/jaehoonpauljeong/Data-Modeling-Group-3-Project. The demonstration video clip is available at our YouTube link of https://www.youtube.com/watch?v=1B-Al1AHDNo&feature=youtu.be. This system is structured as follows:

A. Network Emulation with Mininet

Mininet is used to emulate a network environment, supporting two common topologies:

- **Ring Topology**: Hosts and switches are arranged in a circular configuration, with links between adjacent nodes and hosts.
- **Mesh Topology**: Hosts and switched are connected with a more interconnected structure, allowing more robust paths and redundancy.

Custom functions in `topology_min.py` are used to set up these topologies, including configuring hosts, switches, and link delays.

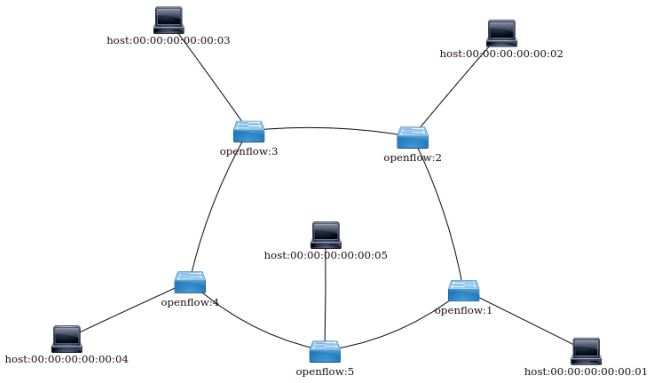

Fig. 1. Ring Topology (implemented with Mininet)

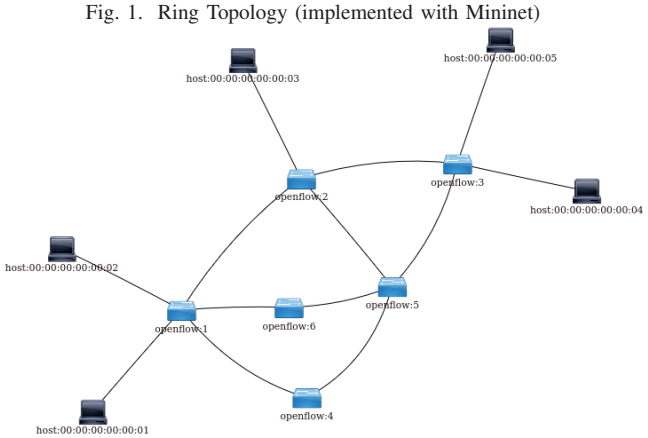

Fig. 2. Mesh Topology (implemented with Mininet)

B. OpenDaylight Controller

The ODL controller is integrated into the Mininet network by its IP address. This allows the centralized control of the switches and other network resources. Communication between the Mininet emulator and the ODL controller is established using either RESTCONF [4] or NETCONF [8], facilitating dynamic topology management and real-time adjustments.

C. Docker Containers

In order to ensure seamless compatibility and ease of deployment, both Mininet and OpenDaylight are hosted within separate Docker containers. This approach ensures a modular architecture, simplifying container management and version control for both components. Additionally, the Python application responsible for implementing dynamic routing is launched within Mininet's container. This design choice consolidates the network emulation and application logic into a single environment, providing an integrated, all-in-one solution for network users. By doing so, the setup reduces complexity, improves portability, and enables easier replication of the environment across different systems.

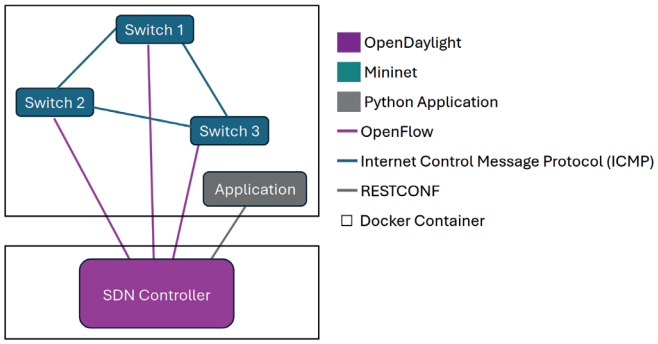

Fig. 3. Visualization of the architecture

D. Network Monitoring and Optimization

- **Latency measurement**: The system monitors network performance using a custom function in `latency.py` that is the implementation of Algorithm 1. Hosts within the Mininet topology perform ping tests, and latency values are extracted and displayed for analysis.
- **Topology and path management**: The Python code in the file `topology_odl.py` interacts with the ODL controller to fetch the current network topology. A graph representation is created using NetworkX to calculate the shortest path between a pair of hosts.
- **Path rerouting**: Based on network performance and congestion, a path between a pair of hosts can be optimized. The `flow.py` module handles path adjustments, allowing the system to propose and apply new routes.

Algorithm 1 Latency Measurement between Hosts in a Network

Input: Network object net containing a list of hosts
Output: Latency measurements for each pair of hosts
$hosts \leftarrow net.hosts$ {Retrieve all hosts from the network}
src in $hosts$, dst in $hosts$ after src
while True **do**
$result \leftarrow$ PingCommand(src, dst) {Run a single ping from src to dst}
$match \leftarrow$ ExtractLatency$(result)$ {Search for a latency value in a ping result}
  if $match$ exists **then** $latency \leftarrow$ extract a numeric value from $match$
break
    if User presses Enter **then** {Check for user interruption}
Print: "Latency measurement stopped by user."
return
Print: "src failed to ping dst, retrying... (press Enter to exit)"
Print: "Latency from src to dst: $latency$ msec"

E. User Interaction

The system provides a command-line interface for user interaction. Through this interface, a user can:

1) Select the desired topology (ring or mesh).
2) Measure latency between a pair of hosts.
3) View and optimize the path between the selected pair.
4) Access the Mininet console for further experimentation.

F. Technological Stack

The application uses Python and mainly the Mininet, NetworkX, and Requests libraries. RESTCONF is used to communicate with the ODL controller, while YANG models define the configuration schema.

IV. Performance Evaluation

This section describes the emulation environment and the performance results of package rerouting in the ring network in Fig. 1 and the mesh network in Fig. 3.

A. Emulation Setup

OpenDaylight and Mininet are launched. The Mesh topology is selected. The Latency Measurement function in Algorithm 1 is executed. Then the optimization function is run on the path between host 2 and host 3. Finally, the latency measurement function is run again and the values are compared.

B. Results

Latency between host 2 and host 3 is higher because the Spanning Tree Protocol (SPT) breaks the ring between the switches ‘openflow:2’ and ‘openflow:3’ although this would be the shortest path between host 2 and host 3 for packets. With the program, new flows have been added to the switches ‘openflow:2’ and ‘openflow:3’ to realize the optimal routing for packets between host 2 and host 3.

	$h1$	$h2$	$h3$	$h4$	$h5$
$h1$	0.000	0.118	0.092	0.116	0.112
$h2$	0.118	0.000	**0.304**	0.102	0.109
$h3$	0.092	**0.304**	0.000	0.126	0.097
$h4$	0.116	0.102	0.126	0.000	0.095
$h5$	0.112	0.109	0.097	0.095	0.000

TABLE I
LATENCY MEASUREMENT MATRIX BEFORE OPTIMIZATION (MSEC)

	$h1$	$h2$	$h3$	$h4$	$h5$
$h1$	0.000	0.121	0.123	0.159	0.147
$h2$	0.121	0.000	**0.152**	0.176	0.176
$h3$	0.123	**0.152**	0.000	0.147	0.177
$h4$	0.159	0.176	0.147	0.000	0.169
$h5$	0.147	0.176	0.177	0.169	0.000

TABLE II
LATENCY MEASUREMENT MATRIX AFTER OPTIMIZATION (MSEC)

V. Conclusion

This paper proposes an SDN Traffic Optimization (SDN-TO) System to quickly optimize traffic flows between hosts through traffic rerouting. As future work, we will explore a dynamic and automated way of traffic rerouting. Also, machine learning techniques will be investigated to further optimize decision-making and scalability.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Ministry of Science and ICT (MSIT), South Korea (No. RS-2024-00398199 and RS-2022-II221199). Note that Jaehoon (Paul) Jeong is the corresponding author.

References

[1] M. Boucadair and C. Jacquenet, “Software-Defined Networking: A Perspective from within a Service Provider Environment,” *RFC 7149*, Mar. 2014. [Online]. Available: https://datatracker.ietf.org/doc/rfc7149/
[2] “Opendaylight.” [Online]. Available: https://www.opendaylight.org/
[3] “Mininet.” [Online]. Available: https://mininet.org/
[4] A. Bierman, M. Bjorklund, and K. Watsen, “RESTCONF Protocol,” RFC 8040, IETF, January 2017. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8040
[5] M. Bjorklund, “YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF),” RFC 6020, IETF, October 2010. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6020
[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling innovation in campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
[7] O. N. Foundation, “OpenFlow Switch Specification - Version 1.5.1,” *ONF TS-025*, Mar. 2015. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf
[8] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, “Network Configuration Protocol (NETCONF),” RFC 6241, IETF, June 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6241
[9] E. C. Rosen, A. Viswanathan, and R. Callon, “Multiprotocol Label Switching Architecture,” *RFC 3031*, Jan. 2001. [Online]. Available: https://datatracker.ietf.org/doc/rfc3031/
[10] D. J. Smith, J. Mullooly, W. Jaeger, and T. Scholl, “Label Edge Router Forwarding of IPv4 Option Packets,” *RFC 6178*, Mar. 2011. [Online]. Available: https://datatracker.ietf.org/doc/rfc6178/
[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT press, 2009.