A Data Model-Driven Load-Balancing Scheme in Software-Defined Networking

Tom Vasseur*, Zulfiyya Gurbanova†, Taegon Park†, Tejas Mhadgut‡, Jaehoon (Paul) Jeong†, and Tae (Tom) Oh‡
*Epita, †Sungkyunkwan University, ‡Rochester Institute of Technology
Email: tom.vasseur@epita.fr, q.zulfiyya.04@gmail.com, taegon1212@g.skku.edu, {tm3886, thoics}@rit.edu, pauljeong@skku.edu

***Abstract*—Traffic congestion remains a significant challenge for resource utilization in modern networks. This paper proposes and implements a dynamic load-balancing system for private Software-Defined Networking (SDN) deployments. Our approach consists of a lightweight Python application that integrates with an OpenDaylight controller to facilitate automated traffic management. It operates in a closed-loop load-balancing control through periodic network traffic analysis. Upon detecting a traffic overload, our SDN system immediately enforces traffic redirection by dynamically modifying OpenFlow rules.**

***Index Terms*—Software-Defined Networking, Load Balancing, OpenDaylight, RESTCONF, YANG, Network Automation.**

I. Introduction

In the era of massive data consumption, the primary challenge for network administrators is no longer just connectivity, but intelligent resource orchestration. Static networking architectures often fail to cope with bursty traffic patterns, leading to severe bottlenecks where specific servers are overwhelmed while others remain idle. Software-Defined Networking (SDN) [1] addresses this inefficiency by centralizing control logic, enabling programmable traffic engineering that is not possible in traditional hardware-centric networks [2].

However, simply deploying an SDN controller is insufficient for optimal performance. Many existing solutions rely on reactive OpenFlow rules, which can introduce latency as the controller processes every new flow request individually. A more sophisticated approach involves proactive network management, in which the network's state is continuously monitored and optimized. This requires leveraging standard Northbound interfaces like RESTCONF and data modeling languages like YANG, which allow external applications to interact with the network's logical structure rather than just its packet-forwarding tables [3].

In this paper, we propose a novel, application-driven load-balancing framework built on the OpenDaylight (ODL) controller. Unlike internal controller modules or static scripts, our design utilizes an external Python-based orchestration engine that establishes a feedback loop with the network. This application queries real-time statistics via RESTCONF, evaluates server load against pre-defined threshold values, and dynamically pushes updated flow rules to redistribute traffic only when necessary. We implemented this logic in a Mininet [4] emulation environment to demonstrate that a threshold-based external control loop can significantly reduce server congestion and improve overall response times compared to static distribution methods.

II. Design

The proposed system is implemented by leveraging the topology information provided by the SDN controller and continuously collecting network traffic statistics (e.g., incoming, forwarded, and dropped packets) from each server-facing OpenFlow port through RESTCONF. Fig. 1 shows a framework of a Load Balancing System. Based on the network traffic statistics, the system computes a static load percentage using a predefined load calculation model. It determines the congestion level of each server by comparing the calculated value against a fixed threshold. Whenever the system detects that a server is overloaded, it dynamically modifies the routing rules to redirect traffic to an alternative server.

The system architecture consists of four main components: (1) Collector, which periodically retrieves port-level traffic statistics from ODL's operational data store; (2) Load Analyzer, which computes the server's network load by deriving throughput and utilization from the collected byte and packet counters; (3) Flow Rule Generator, which constructs OpenFlow flow entries that redirect traffic destined for an overloaded server to a different backend; (4) Dynamic Load Offloading Engine, which evaluates threshold conditions and installs flow rules through RESTCONF to enforce runtime traffic redirection.

The Collector fetches real-time port statistics (e.g., transmitted bytes, received bytes, and dropped packets) at fixed timestamps. In the Load Analyzer, the system computes the utilization by comparing the current and previous measurements and converting the delta values into throughput and percentage-based load. A higher calculated load value indicates a sudden increase in traffic directed toward that server.

When the load exceeds the predefined threshold, the Flow Rule Generator produces a redirection rule that rewrites the destination IP address of incoming flows to an alternative server. The Load Offloading Engine then installs this rule into the OpenFlow flow table by issuing a RESTCONF PUT request to the controller's configuration datastore. Once deployed, the switch immediately begins forwarding traffic to the backup server.

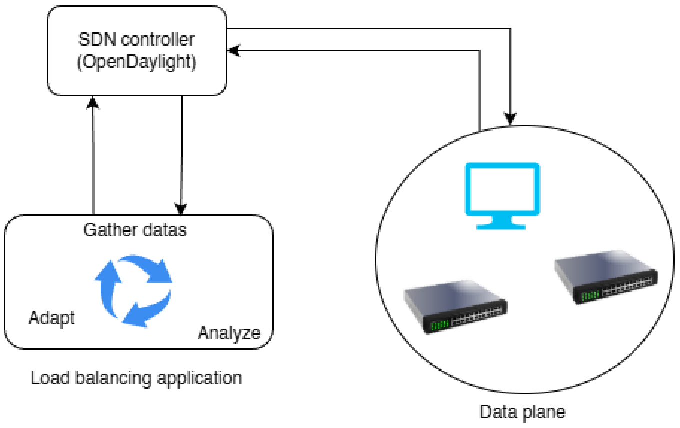


Fig. 1. A Framework of a Load Balancing System

The entire process is implemented in Python and operates continuously, enabling real-time monitoring and dynamic traffic steering with minimal control overhead.

III. Implementation

To validate our proposed load-balancing framework, we developed a comprehensive prototype consisting of three main components: a custom network topology emulated in Mininet, an OpenDaylight (ODL) SDN controller, and an external Python-based orchestration application. This section details the setup and the logic used to enforce dynamic traffic management.

A. *Experimental Environment*

Our experimental setup uses two distinct virtual machines (VMs) to simulate a realistic separation between the control and data planes.

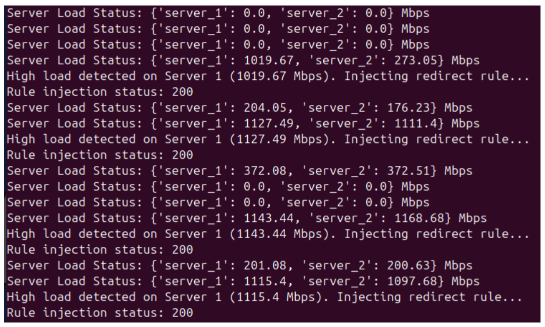
- **VM 1 (Controller):** It hosts the OpenDaylight controller (Chlorine release). It acts as the centralized brain of the network.
- **VM 2 (Data Plane):** It hosts the Mininet emulation environment and our custom Python orchestration scripts.

B. *Traffic Generation and Testing*

To stress-test the implementation, we used *iPerf* to generate traffic from the clients to the servers. The test scenario involved initializing high-bandwidth streams (100 Mbps) from both clients targeting a single server to force a congestion scenario, verifying that our application correctly detects the threshold breach and installs the necessary redirection rules.

IV. Evaluation

To validate the system, we monitored real-time traffic handling through the Python orchestration console. As shown in Fig. 2, the system successfully detects when Server 1 exceeds the defined threshold (reaching 1019.67 Mbps). The application immediately triggers a mitigation routine, confirmed by the "Rule injection status: 200" log entry, which indicates successful communication with the OpenDaylight controller. Subsequent readings show a significant load reduction on Server 1 (dropping to 204.05 Mbps) as traffic is redistributed to Server 2, proving the effectiveness of the automated redirection logic.



```
Server Load Status: {'server_1': 0.0, 'server_2': 0.0} Mbps
Server Load Status: {'server_1': 0.0, 'server_2': 0.0} Mbps
Server Load Status: {'server_1': 0.0, 'server_2': 0.0} Mbps
Server Load Status: {'server_1': 1019.67, 'server_2': 273.05} Mbps
High load detected on Server 1 (1019.67 Mbps). Injecting redirect rule...
Rule injection status: 200
Server Load Status: {'server_1': 204.05, 'server_2': 176.23} Mbps
Server Load Status: {'server_1': 1127.49, 'server_2': 1111.4} Mbps
High load detected on Server 1 (1127.49 Mbps). Injecting redirect rule...
Rule injection status: 200
Server Load Status: {'server_1': 372.08, 'server_2': 372.51} Mbps
Server Load Status: {'server_1': 0.0, 'server_2': 0.0} Mbps
Server Load Status: {'server_1': 0.0, 'server_2': 0.0} Mbps
Server Load Status: {'server_1': 1143.44, 'server_2': 1168.68} Mbps
High load detected on Server 1 (1143.44 Mbps). Injecting redirect rule...
Rule injection status: 200
Server Load Status: {'server_1': 201.08, 'server_2': 200.63} Mbps
Server Load Status: {'server_1': 1115.4, 'server_2': 1097.68} Mbps
High load detected on Server 1 (1115.4 Mbps). Injecting redirect rule...
Rule injection status: 200
```

Fig. 2. The Output of a Python Load Balancing Application

This external application-plane approach offers significant flexibility, allowing administrators to write complex logic in Python without modifying the controller's internal code. However, it introduces latency due to the overhead of HTTP RESTCONF polling compared to native internal modules. Research by Shalimov et al. [5] highlights this trade-off, noting that while Northbound Interfaces ensure interoperability, they are generally slower than internal Southbound interactions for processing high-frequency flow events.

V. Conclusion

This paper presented a hybrid load-balancing framework that integrates an OpenDaylight controller with an external Python-based orchestration engine. By leveraging the RESTCONF/YANG interface, our solution effectively automates traffic monitoring and flow redirection. Experimental results demonstrate that the system enables dynamic resource optimization and congestion relief, offering a programmable and vendor-neutral alternative to static routing configurations.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant (No. RS-2024-00398199) and the National Research Foundation of Korea (NRF) grant (No. 2023R1A2C2002990) funded by the Ministry of Science and ICT (MSIT), South Korea. Jaehoon (Paul) Jeong is the corresponding author.

References

[1] Q. Du, X. Cui, H. Tang, and X. Chen, "Review of load balancing mechanisms in sdn-based data centers," *Journal of Computer and Communications*, vol. 12, no. 1, pp. 49–66, 2024.

[2] M. Hamdan, E. Hassan, A. Abdelaziz, and A. Elhoseny, "Intelligent load balancing techniques in software defined networks: A survey," *Electronics*, vol. 9, no. 7, p. 1107, 2020.

[3] M. D. Tache, O. Pascutoiu, and E. Borcoci, "Optimization algorithms in sdn: Routing, load balancing, and delay optimization," *Applied Sciences*, vol. 14, no. 14, p. 5967, 2024.

[4] Mininet, "Mininet: An instant virtual network on your laptop," 2022, accessed: 2025-05-15. [Online]. Available: http://mininet.org/

[5] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," in *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, 2013, pp. 1–6. [Online]. Available: https://dl.acm.org/doi/10.1145/2556616.2556618