Cluster-Based Detection of Compromised SDN Controller Using Behavioral Divergence

Jiwon Suh*, Jaehoon (Paul) Jeong*, and Tae (Tom) Oh†
*Department of Computer Science & Engineering, Sungkyunkwan University, Suwon, Republic of Korea
Email: {sjw6136, pauljeong}@skku.edu
†School of Information, Rochester Institute of Technology, USA
Email: thoics@rit.edu

***Abstract*—Software-Defined Networking (SDN) enhances programmability by decoupling the control and data planes, yet this centralization introduces a critical vulnerability: a compromised controller can manipulate traffic while remaining protocol-compliant. Existing global monitoring techniques struggle with scalability in large data centers. This paper proposes a scalable anomaly detection framework that leverages traffic locality in hierarchical SDN topologies. By clustering switches into localized monitoring domains, the framework computes behavioral indices per cluster and measures inter-cluster divergence using Max Pairwise Distance (MPD). A supervised learning model classifies these patterns to detect anomalies without requiring full-network visibility. Using DCT2Gen for realistic traffic generation, our method demonstrates high detection accuracy with low overhead, making it suitable for large-scale SDN deployments.**

***Index Terms*—Software-Defined Networking, Compromised Controller Detection, Cluster-Based Modeling, Data Center Networks, Max Pairwise Distance, Network Security.**

I. Introduction

Software-Defined Networking (SDN) decouples the control plane from the data plane, enabling centralized, programmable control over network behavior [1]. This architecture offers benefits such as global visibility, streamlined policy enforcement, and simplified management—key for hyperscale data centers using Clos or Fat-Tree topologies to support dynamic, spatially localized traffic [2]–[5].

However, centralization also introduces a critical vulnerability: the controller becomes a single point of failure. If compromised, it can manipulate flow rules or disrupt traffic without violating protocols [6]–[8]. Existing detection methods rely on global traffic statistics or centralized Machine Learning (ML), which do not scale well and often miss local anomalies [9].

We propose a scalable detection framework that leverages the hierarchical structure of data center networks, which distributes the monitoring tasks across clusters instead of concentrating them in a centralized system. Switches are clustered into pods (i.e., clusters), and anomalies are detected by measuring inter-cluster behavioral divergence using Max Pairwise Distance (MPD). This decentralized approach reduces central monitoring overhead and improves sensitivity to local threats.

Our contributions are as follows:

- Identify key limitations of an existing SDN controller's anomaly detection in scalability and spatial awareness.
- Propose an MPD-based detection framework using pod-level clustering.
- Demonstrate high detection accuracy with low central monitoring overhead via emulation.

The implementation is available online as an open-source repository [10].

The remainder of this paper is structured as follows. Section II reviews related work. Section III describes the system model. Section IV outlines the threat model. Section V details the behavioral index. Section VI discusses the emulation setup. Section VII presents performance evaluation, and Section VIII concludes this paper along with future work.

II. Related Work

Anomaly detection in SDN has been widely studied due to the controller's central role. Prior approaches fall into three categories: global traffic analysis, local detection, and topology-aware monitoring.

Global monitoring uses centralized metrics like flow volume or control-message frequency to detect anomalies [6], [9]. While effective in small-scale settings, these methods assume full visibility and stable traffic, making them less suitable for large, dynamic data centers. They also incur high overhead [11].

Local detection reduces overhead by observing individual switches or distributing analysis [12], [13]. BDF-SDN [12] applies big-data analysis for the detection of

978-4-907626-55-6/25/$31.00 ©2025 IPSJ

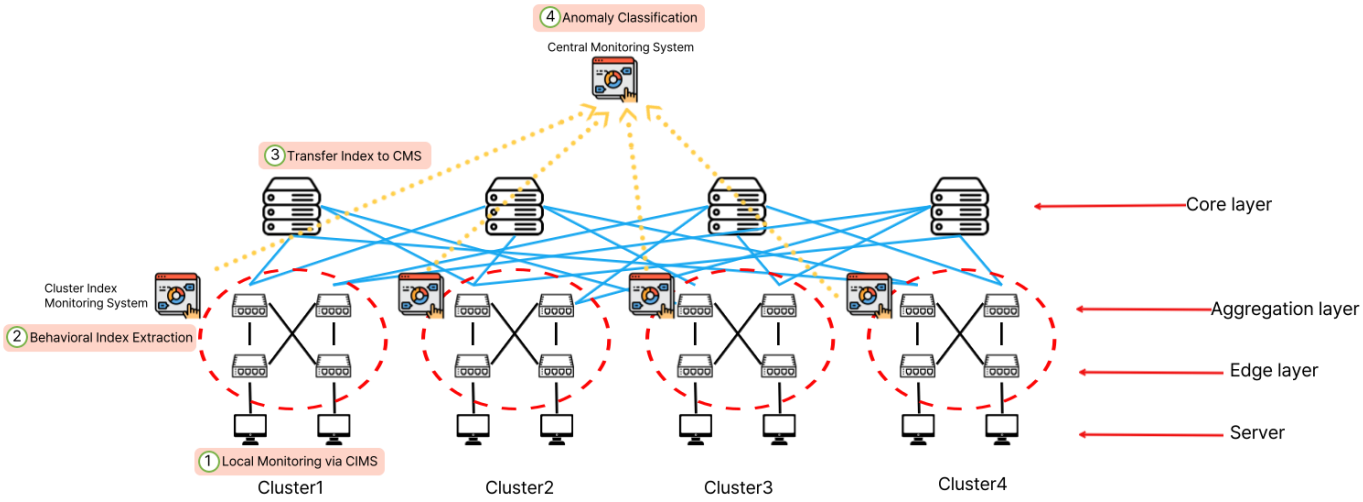

Fig. 1. Framework for compromised controller detection using pod-based Cluster Index Monitoring.

Distributed Denial of Service (DDoS), but often requires fine-grained traces or controller introspection, limiting scalability [14].

Topology-aware methods like TopoGuard+ [15] and Popic et al. [16] detect routing anomalies by monitoring control-plane topology. These approaches validate topology integrity but do not capture data-plane behavioral anomalies.

Some works address SDN efficiency via clustering or flow aggregation. Yeung and Wong [17] in cluster switches to optimize controller placement, while RFC 8889 [18] and RFC 9342 [19] improve multicast handling and load balancing. These methods focus on control efficiency rather than security.

In contrast, our method leverages clustering to detect behavioral anomalies by analyzing inter-cluster divergence. It operates without full network visibility and is well, suited for hierarchical SDN architectures, where spatial locality is prevalent.

III. System Model

A. Data Center Topology Overview

Modern data centers typically use hierarchical multi-tier topologies such as FatTree and Clos to ensure scalability, fault tolerance, and efficient traffic routing. These networks are divided into four logical layers:

- **Core layer**: Interconnects pods via high-bandwidth links.
- **Aggregation layer**: Connects edge switches to the core and supports intra-pod traffic.
- **Edge layer**: Interfaces with servers.
- **Server layer**: Hosts compute nodes generating traffic.

Pods, which group edge and aggregation switches, serve as independent traffic-locality units. Since east-west traffic is primarily confined within pods, we adopt a pod-based clustering model for behavioral monitoring.

B. Proposed Cluster Index Monitoring Framework

Our decentralized monitoring framework leverages traffic locality. Each cluster corresponding to a pod includes a local agent called the Cluster Index Monitoring System (CIMS).

CIMS collects local switch statistics and computes behavioral indices that capture control-plane behavior, including flow rule distribution, parameter frequency, and packet handling.

These indices are sent to a central system called Cental Monitoring System (CMS) that measures inter-cluster divergence using Max Pairwise Distance (MPD):

$$\mathrm{MPD}(C) = \max_{i,j \in C, i \neq j} |\mathbf{B}_a - \mathbf{B}_p|_2, \tag{1}$$

where $\mathbf{B}_a$ and $\mathbf{B}_p$ are behavioral index vectors for clusters i and j. For (1), normal conditions yield low MPD due to a similar behavior. Divergence indicates potential anomalies.

CIMS Framework's advantages:

- **Scalability**: Each CIMS processes local data.
- **Accuracy**: Intra-pod consistency enables fine-grained anomaly detection.
- **Efficiency**: Summary vectors reduce central monitoring overhead.

- **Resilience**: Detection contimues despite partial CIMS failure.

By exploiting the natural structure and locality of modern data centers, our framework provides an efficient and robust mechanism for detecting compromised SDN controller behavior in large-scale environments.

IV. Threat Model

We assume a threat model in which a single SDN controller, while remaining OpenFlow-compliant, is compromised and issues malicious flow rules that subtly alter traffic without violating protocols.

The attacker may execute the following actions:

- **Vector 1: Path Concentration** – Force traffic through fixed switches to create bottlenecks.
- **Vector 2: Stealthy Mirroring** – Duplicate packets to covert hosts while preserving normal delivery.
- **Vector 3: Silent Drops** – Drop packets randomly at low rates to degrade Quality of Service (QoS).
- **Vector 4: Resource Exhaustion** – Overload switches via excessive flow requests.
- **Vector 5: Asymmetric Forwarding** – Introduce one-way rerouting or mirroring to create inconsistencies.

The following vectors form three controller compromise types:

- **Type I: Path Manipulation** – Uses Vectors 1 and 4 to alter routing or create hotspots.
- **Type II: Stealthy Mirroring** – Uses Vectors 2 and 5 to duplicate or misroute packets.
- **Type III: Silent Drop** – Uses Vector 3 to probabilistically degrade performance.

V. Behavioral Index Design

To identify behavioral deviations induced by compromised controllers, we define a set of five statistical metrics derived from flow table contents and packet-level activities per switch. These metrics are computed independently per cluster and later compared using the MPD defined in Section III.

A. Switch Participation Index (SPI)

Definition: Measures the variance in flow rule distribution across switches, normalized by port count. It reflects how evenly traffic is spread based on switch connectivity.

Formula:

$$\text{SPI} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i}{d_i} - \mu\right)^2, \quad \mu = \frac{1}{N}\sum_{i=1}^{N}\frac{x_i}{d_i}, \tag{2}$$

- x_i: Number of flow rules on switch i
- d_i: Degree (ports) of switch i
- N: Total number of switches

B. Priority Frequency Spike Index (PFSI)

Definition: Measures the imbalance in the use of priority values across flow rules. Skewed distributions may suggest manipulation.

Formula:

$$\text{PFSI} = \max\left(\max(n_i) - \mu,\ \mu - \min(n_i)\right), \mu = \frac{1}{P}\sum_{i=1}^{P} n_i, \tag{3}$$

- n_i: Number of flow rules with priority level i
- P: Number of distinct priority levels observed

C. Timeout Frequency Spike Index (TFSI)

Definition: Detects irregularities in idle_timeout configurations among flow rules, which can reflect control anomalies.

Formula:

$$\text{TFSI} = \max\left(\max(t_i) - \mu,\ \mu - \min(t_i)\right), \mu = \frac{1}{T}\sum_{i=1}^{T} t_i, \tag{4}$$

- t_i: Number of flow rules with timeout value i
- T: Number of distinct timeout values observed

D. Packet Processing Ratio (PPR)

Definition: Compares outgoing and incoming packets to assess forwarding success within a cluster.

Formula:

$$\text{PPR} = \frac{\sum_{i=1}^{N} \text{packet_out}_i}{\sum_{i=1}^{N} \text{packet_in}_i}, \tag{5}$$

- packet_in_i: Number of PacketIn events at switch i
- packet_out_i: Number of PacketOut events from switch i

E. Packet Processing Divergence (PPD)

Definition: Detects excessive outgoing packets relative to incoming ones—suggesting duplication or unauthorized mirroring.

Formula:

$$\text{PPD} = \begin{cases} \frac{\sum \text{packet_out} - \sum \text{packet_in}}{\sum \text{packet_in}}, & \text{if } \sum \text{packet_out} > \sum \text{packet_in}, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

These five metrics form the core of our cluster-based detection framework. They provide compact summaries of behavioral patterns and enable scalable anomaly detection across distributed monitoring domains.

F. Threat-to-Index Mapping Overview

TABLE I
THREAT VECTORS AND CORRESPONDING DETECTION INDEXES

ID	Threat Description	Type	Indexes
V1	Forced waypoint routing; traffic imbalance.	I	SPI, PFSI
V2	Covert packet duplication for exfiltration.	II	PPD, PFSI
V3	Silent packet drops degrading QoS.	III	PPR, SPI
V4	Flow rule overload to exhaust resources.	I	TFSI, SPI
V5	Asymmetric routing or mirroring.	II	PPD, TFSI

Table I summarizes the relationships between the defined threat vectors, the implemented controller types, and the behavioral indexes most effective at detecting each attack pattern.

G. Behavioral Similarity Across Clusters

In benign settings, hierarchical SDN networks configured with uniform controller logic produce highly similar behavioral indexes across clusters. Our emulations confirm that MPD remains low under normal conditions and increases significantly under controller compromise.

Figs. 2 and 3 illustrate this trend using 50 representative samples per class across 20 switches. In both cases, benign clusters show tightly grouped MPD values, while compromised clusters exhibit clear divergence.

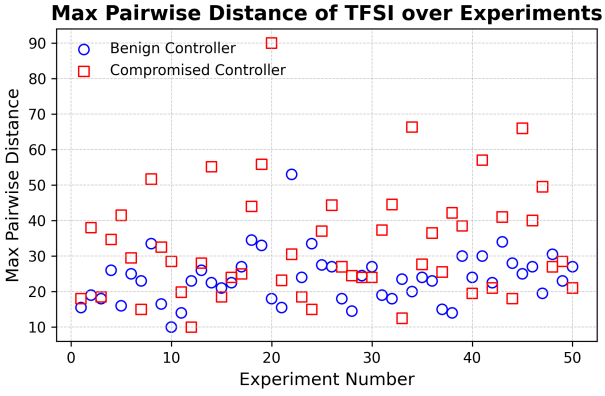

Fig. 2. PFSI-Based MPD by Condition

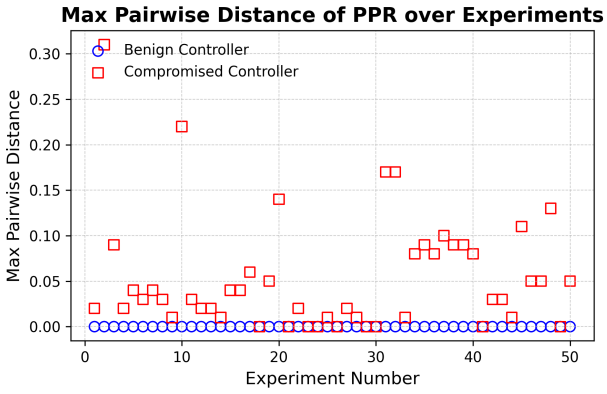

Fig. 3. PPR-Based MPD by Condition

VI. EMULATION SETUP

A. Infrastructure Overview

We emulate our SDN-based data center using Mininet and Ryu on a virtualized Linux environment. Key specifications are summarized in Table II.

TABLE II
EMULATION INFRASTRUCTURE CONFIGURATION

Item	Description
Network Emulator	Mininet v2.3.1b4
SDN Controller	Ryu v4.34
Operating System	Ubuntu 20.04 (VM)
Virtualization	VMware Workstation
CPU	8-core processor
Memory	32 GB
Disk Space	50 GB

B. Network Topology

We adopt a k-ary Fat-Tree topology generator [20] to emulate scalable hierarchical networks. Topology size scales with k using standard formulas [21]. A summary is provided in Table III.

TABLE III
FAT-TREE TOPOLOGY SUMMARY FOR $k = 4, 6, 8, 10$ (BASED ON IMPLEMENTATION)

k	Clusters	Core	Agg.	Edge	Total Switches	Hosts
4	4	4	8	8	20	16
6	6	9	18	18	45	36
8	8	16	32	32	80	64
10	10	25	50	50	125	100

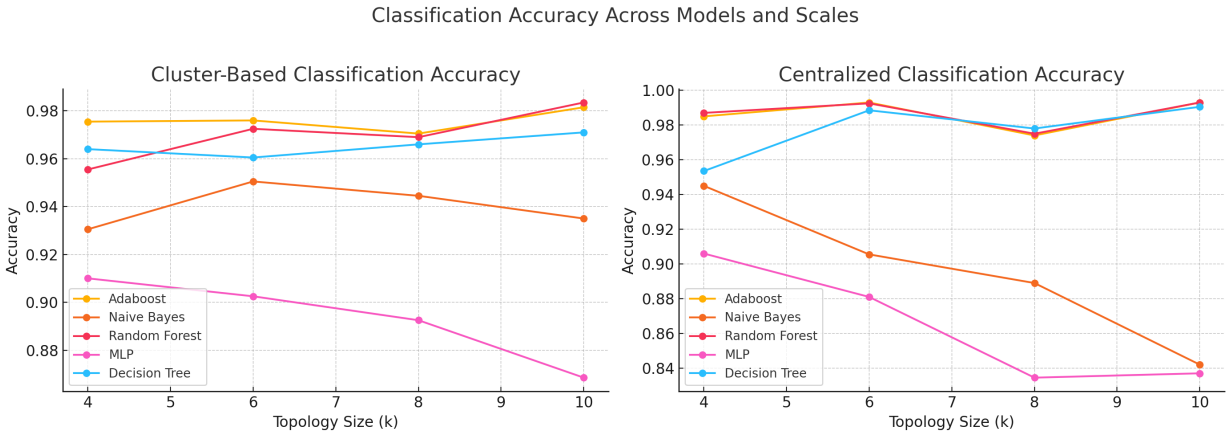

Fig. 4. Classification Accuracy Across Models and Topology Sizes.

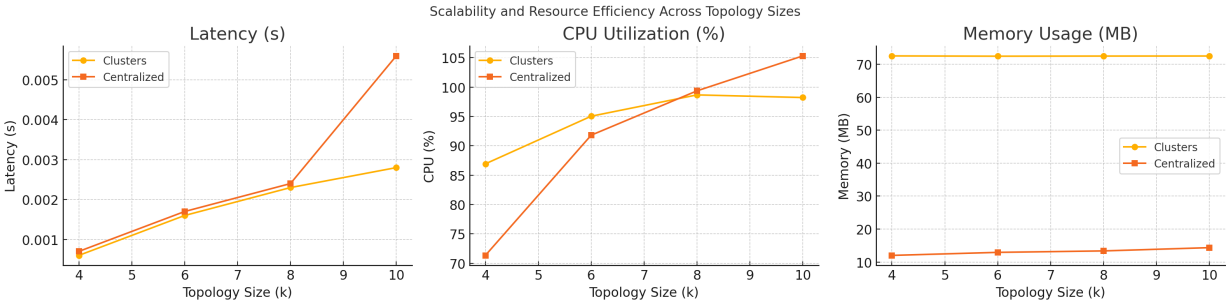

Fig. 5. Scalability and Resource Efficiency Comparison.

C. Traffic Execution and Behavioral Monitoring

We generate synthetic traffic using a DCT2Gen [22]-inspired tool, emulating data center patterns with burstiness, spatial locality, and variable flow sizes. Each emulation runs 30 seconds; random host pairs exchange log-normal flows with exponential inter-arrival times. Identical traffic was applied within a trial for consistency.

D. Attack Scenario Generation

To generate compromised samples, we implemented the three attack types described in Section IV by modifying the Ryu controller. For Type I, traffic was rerouted through fixed intermediate switches to emulate congestion. Type II added flow rules that mirrored packets to unauthorized ports. Type III introduced low-probability packet drops via special rules.

These behaviors were selectively applied to clusters during emulation, and the resulting flow statistics were used to compute behavioral indexes for classifier training and evaluation.

E. Classifier-Based Detection Evaluation

To evaluate detection performance, we extract behavioral indexes in two formats: aggregated values across the entire network and per-cluster values used for MPD computation. For each topology size, we construct a balanced dataset of 5,000 benign and 5,000 compromised training samples, and a separate test set of 1,000 samples per class.

We compare five classifiers—Naïve Bayes, Decision Tree, Random Forest, Adaboost, and Multi-Layer Perceptron (MLP)—under centralized and cluster-based detection modes according to the two input formats described above.

F. Scalability and Resource Efficiency Evaluation

To assess the practical feasibility of our approach, we compare the centralized and cluster-based detection methods with respect to the computational load on the CMS, measured in CPU usage, memory overhead, and processing latency. Each metric is averaged over 10,000 training instances per topology size.

VII. Performance Evaluation

A. Classification Performance

Fig. 4 compares classification accuracy across five models under both the centralized and cluster-based detection methods. Cluster-based detection method shows

more stable accuracy as k increases, while the centralized methods—particularly Naïve Bayes and MLP—exhibit higher variability, suggesting that localized behavioral patterns are more robust at scales (i.e., topology size).

B. Scalability and Resource Efficiency

Fig. 5 summarizes latency, CPU usage, and memory overhead. The cluster-based method achieves lower latency and stable memory usage, especially at larger topologies. While CPU usage grows with network size in both approaches, it becomes steady for cluster-based detection, showing better scalability.

These results confirm that our framework combines high accuracy with efficient central monitoring resource usage.

VIII. Conclusion

We presented a scalable framework for detecting compromised SDN controllers by leveraging traffic locality and cluster-based behavioral monitoring with low overhead for large-scale environments. While our evaluation demonstrates strong scalability and accuracy, it has some limitations. First, the approach assumes that at least some clusters remain unaffected; if all clusters are compromised identically, MPD may remain low and hinder detection. Second, all emulations were conducted in an emulated environment with a single-controller architecture, which does not fully capture the complexity of real-world deployments.

As future work, we will incorporate temporal analysis within clusters, compare against historical baselines, and extend the framework to support multi-controller SDN environments. We also aim to validate our approach on real hardware testbeds to assess its robustness under practical conditions.

Acknowledgments

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

References

[1] 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 Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[2] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM*, 2009, pp. 51–62.

[3] R. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, V. Radhakrishnan, A. V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *Proc. ACM SIGCOMM*, 2009, pp. 39–50.

[4] N. Farrington and A. Andreyev, "Facebook's data center network architecture," in *Proc. Optical Interconnects Conf.* IEEE, 2013, pp. 49–50.

[5] M. Polák, D. Sedlák, J. Fesl, and P. Tvrdík, "Real data center network traffic dataset and analysis," in *Proc. IEEE Int. Conf. Cloud Networking (CloudNet)*, 2024.

[6] A. Anand *et al.*, "On detecting compromised controller in sdn," in *Proc. Int. Conf. Adv. Comput., Commun. Inform. (ICACCI)*. IEEE, 2018, pp. 2342–2347.

[7] A. Sinha *et al.*, "Ddos vulnerabilities analysis in sdn controllers: Understanding the attacking strategies," *Int. J. Commun. Syst.*, 2023.

[8] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[9] T. Phu *et al.*, "Defending sdn against packet injection attacks using deep learning," in *Proc. Int. Conf. Information Networking (ICOIN)*. IEEE, 2023, pp. 500–505.

[10] J. Suh, "Cluster-based detection of compromised sdn controller," GitHub Repository, available at: github.com/Seo6136/Cluster-Based-Detection-of-Compromised-SDN-Controller-Using-Behavioral-Divergence.

[11] D. Abts and B. Felderman, "The cost of a cloud: Research problems in data center networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, 2012.

[12] Y. Zhang *et al.*, "Bdf-sdn: A big data framework for ddos attack detection in large-scale sdn-based cloud," *J. Netw. Comput. Appl.*, vol. 68, pp. 45–54, 2016.

[13] K. DeLany *et al.*, "Sdn dynamic controller configuration to mitigate compromised controllers," in *Proc. IEEE Conf. Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2023.

[14] A. Tootoonchian and Y. Ganjali, "Opennetmon: Network monitoring in openflow software-defined networks," in *Proc. ACM CoNEXT*, 2012, pp. 109–120.

[15] X. Wu, L. Xu, and G. Gu, "Effective topology tampering attacks and defenses in software-defined networks," in *Proc. IEEE/IFIP DSN*, 2018, pp. 161–172.

[16] S. Popic, M. Vuleta, P. Cvjetkovic, and B. M. Todorovic, "Secure topology detection in software-defined networking with network configuration protocol and link layer discovery protocol," in *Proc. TELFOR*. IEEE, 2020, pp. 1–4.

[17] L. K. Yeung and T. Wong, "Cluster-based control in software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1481–1495, 2019.

[18] M. Abrahamsson and H. Alvestrand, "Igmp and mld proxying," RFC 8889, 2020.

[19] T. Mizrahi and M. Spiegel, "Flow aggregation for sdn load balancing," RFC 9342, 2023.

[20] H. M. Khan, "Fat-tree-data-center-topology," GitHub Repository, available at: github.com/HassanMahmoodKhan/Fat-Tree-Data-Center-Topology.

[21] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM*, 2008, pp. 63–74.

[22] P. Wette and H. Karl, "Dct2gen: A traffic generator for data centers," *Comput. Commun.*, vol. 80, pp. 45–58, 2016.