Agent-Based End-Host Monitoring in IPv6 SDN Intranets: A Hybrid Machine Learning Approach by using UEBA Framework

Bishal Panta, Arbashu Dhakal, Adhyadesh Dahal
Department of Electronics and Computer Engineering
Pulchowk Campus, IOE, TU, Lalitpur, Nepal
Emails: {078bct036.bishal, 078bct018.arbashu, 078bct007.adhyadesh}@pcampus.edu.np

Babu R. Dawadi
Department of Electronics and Computer Engineering
Pulchowk Campus, IOE, TU, Lalitpur, Nepal
Email: baburd@ioe.edu.np

Bishnu Prasad Gautam
Department of Applied Information Engineering
Suwa University of Science
Nagano, Japan
Email: gautam_bishnu@rs.sus.ac.jp

Roshani Ghimire
Department of Electronics and Computer Engineering
Advanced College of Engineering and Management
IOE, TU, Kathmandu, Nepal
Email: roshanigd@gmail.com

***Abstract*—SDN architectures prioritize centralized network control but often lack visibility into host-level activity, resulting in blind spots for insider threats and stealthy attacks. A key challenge lies in developing a scalable, secure, and low-overhead host-monitoring solution that integrates seamlessly with SDN infrastructures. This paper addresses that gap by proposing an agent-based end-host monitoring architecture designed for security-critical SDN-enabled IPv6 intranets. Our method involves deploying lightweight monitoring agents on each host, which collect system and process metrics as time-series data and transmit them securely to a centralized analytics engine via mTLS-protected gRPC streams. The core analytical framework relies on User and Entity Behavior Analytics (UEBA), a behavior-based cybersecurity paradigm that establishes multidimensional behavioral baselines for users, machines, and processes. UEBA employs machine learning techniques to detect deviations from these baselines, capturing both abrupt and subtle anomalies. In our implementation, the system adopts a hybrid detection pipeline that combines the isolation forest and the LSTM autoencoders. Detected anomalies are relayed to the SDN controller through its northbound APIs, enabling dynamic enforcement of flow rules and real-time response. Evaluation in a controlled testbed confirms that the system maintains low CPU and memory overhead per agent while achieving F1-scores of 0.70 and 0.799 for hardware and software-level anomalies, respectively. With sub-second telemetry latencies and near-real-time anomaly signaling, the proposed architecture demonstrates practical feasibility for deployment in security-critical SDN-managed environments.**

***Index Terms*—SDN, End Host Monitoring, Anomaly Detection, UEBA**

I. Introduction

Software-Defined Networking (SDN) has transformed network management by decoupling the control plane from the data plane, enabling centralized control, automation, and dynamic resource allocation. While SDN enhances network programmability and efficiency, it also introduces security challenges, particularly in environments handling sensitive data and critical services [1]. Traditional SDN security mechanisms primarily focus on monitoring network infrastructure components such as switches and controllers [2]. However, this network-centric approach neglects threats that originate or manifest within end-hosts. These include malware operating entirely within a host, insider misuse of credentials, and stealthy data exfiltration techniques. These threats remain invisible because SDN controllers lack direct introspection into host system metrics or process activity, and existing telemetry systems do not feed into the SDN policy loop. This creates a security blind spot, which we define as the inability of SDN controllers to observe and respond to host-internal behaviors that do not manifest as anomalous flows. Furthermore, limited integration between host-level data and SDN control logic makes it difficult to correlate local host behavior with network-wide policies. Despite increasing SDN adoption in enterprise and critical infrastructure, robust host-level monitoring mechanisms remain underexplored.

Bridging this gap requires a system that can collect fine-grained host telemetry, detect anomalies in real time, and translate these findings into actionable SDN policies without compromising performance or scalability. To address these challenges, this work introduces an agent-based monitoring architecture that integrates host-level telemetry and UEBA-driven machine learning-based anomaly detection with SDN's centralized policy control. The proposed system deploys lightweight agents on each host to collect granular system metrics. These metrics are securely transmitted over mutual TLS (mTLS) via gRPC to a centralized analytics engine,

This is the author's accepted manuscript of a paper published by IEEE. The final published version is available at https://doi.org/10.1109/NaNA66698.2025.00012.

which performs behavior modeling and anomaly detection using User and Entity Behavior Analytics (UEBA). UEBA is a behavioral security approach that moves beyond signature-based detection by establishing baseline activity patterns for users, processes, and machines, then identifying statistically or temporally abnormal deviations from those baselines.

To realize this integration, our system adopts a hybrid detection pipeline that combines Isolation Forests for rapid detection of hardware-level outliers and LSTM-based autoencoders for modeling process-level behavior over time. This architecture balances runtime efficiency and temporal precision, enabling timely yet nuanced detection of anomalous activity. The system is designed to operate securely and efficiently within SDN infrastructures, allowing host-level alerts to trigger network-level policy enforcement through northbound APIs exposed by the SDN controller.

The main contributions of this paper are: (i) a secure, agent-based host monitoring architecture for IPv6 SDN intranets; (ii) a hybrid UEBA-based anomaly detection pipeline combining Isolation Forests and LSTM Autoencoders; (iii) dynamic SDN flow rule enforcement triggered by host-level detection; and (iv) empirical validation via anomaly injection tests, evaluating detection accuracy, overhead, and scalability.

The remainder of this paper is organized as follows. Section II reviews related work in SDN security and host-based anomaly detection. Section III outlines the architecture and implementation. Section IV presents evaluation results. Section V discusses limitations, scalability, and deployment considerations. Finally, Section VI concludes the paper and outlines future research directions.

II. Related Works

Authors in [3] developed an End-host Driven Troubleshooting Architecture (EDT) for SDN environments. This system leveraged end hosts to share application-specific connection details with the network infrastructure, improving root-cause identification for performance issues. EDT introduced techniques such as hop-by-hop latency measurements (HEL) and packet tracing (PTR), significantly reducing control plane overhead and troubleshooting times compared to conventional approaches.

One notable host-based SDN approach is DeepContext [4], which utilizes the host machine itself as a switch to extract end-host information. While this OpenFlow-compatible approach allowed for data collection and communication with a centralized controller, it faces scalability challenges due to the overhead introduced on host systems.

Authors in [5] introduced a Software-defined Unified Monitoring Agent (SUMA) that addresses network management challenges in SDN by providing an intelligent middleware for monitoring and filtering network data. While focusing on switch-side monitoring, SUMA complements end-host monitoring approaches by offering a centralized management abstraction that reduces control traffic overhead and enhances network visibility. The work provides insights into integrated monitoring strategies that can inform comprehensive security monitoring in SDN environments. Authors in [6] proposed ZT-SDN, a zero-trust SDN framework using unsupervised learning on transactional graphs to detect and mitigate abnormal flow behavior.

In the context of cloud, authors in [7] proposed V-Sight, a monitoring framework designed for virtual networks in clouds. V-Sight addresses key challenges in cloud environments such as non-isolated statistics and excessive control plane overhead by introducing mechanisms for transmission disaggregation. Although focused on cloud settings, the design principles offer relevant strategies for scalable telemetry collection in SDN-managed systems.

To enhance anomaly detection capabilities, machine learning techniques have been widely adopted in network security. User and Entity Behavior Analytics (UEBA) methods are particularly relevant, spanning malware identification, insider threat detection, and data leakage monitoring, often leveraging system call traces or behavioral logs as input data [8]. For instance, authors in [9] described a UEBA platform that relies on Singular Value Decomposition (SVD) to establish behavioral baselines and identify deviations. However, their system is tailored to centralized enterprise servers and uses both historical and peer-group behavior modeling. Authors in [10] applied K-means algorithm and graph analysis to UEBA output using multiplex networks.

In contrast, our work targets fine-grained, per-host monitoring in intranet-scale deployments and eliminates reliance on peer-group baselines. We further diverge by employing a hybrid pipeline that combines Isolation Forest for real-time detection of hardware-level anomalies and LSTM-based autoencoders for temporal modeling of software behavior. Moreover, unlike [9], which does not explicitly support SDN policy integration, our system is tightly coupled with an SDN controller to translate detected anomalies into real-time mitigation actions.

Deep learning-based anomaly detection has also gained traction in the context of host or network telemetry. Convolutional Autoencoders (CAEs) and LSTM-based Autoencoders have been used for learning compact latent representations of time series data and detecting anomalies via reconstruction loss [11]–[13]. These models have shown strong performance in capturing nonlinear temporal dynamics in system behavior. In our architecture, the LSTM Autoencoder is used specifically to learn normal process behavior across short time windows, with anomaly decisions made based on reconstruction error.

Likewise, unsupervised techniques like Isolation Forest have shown efficacy in low-overhead, high-speed anomaly detection pipelines. Prior works [14], [15] demonstrate its utility on traffic-level features, particularly in scenarios where labeling is impractical. Our implementation adapts iForest to monitor system-level metrics such as CPU, memory, and I/O utilization, providing an efficient method for spotting behavioral outliers at the system level.

Recent surveys in time-series anomaly detection (TSAD) have introduced more expressive architectures, including DAEMON [16], DCT-GAN [17], and Interfusion, which aim

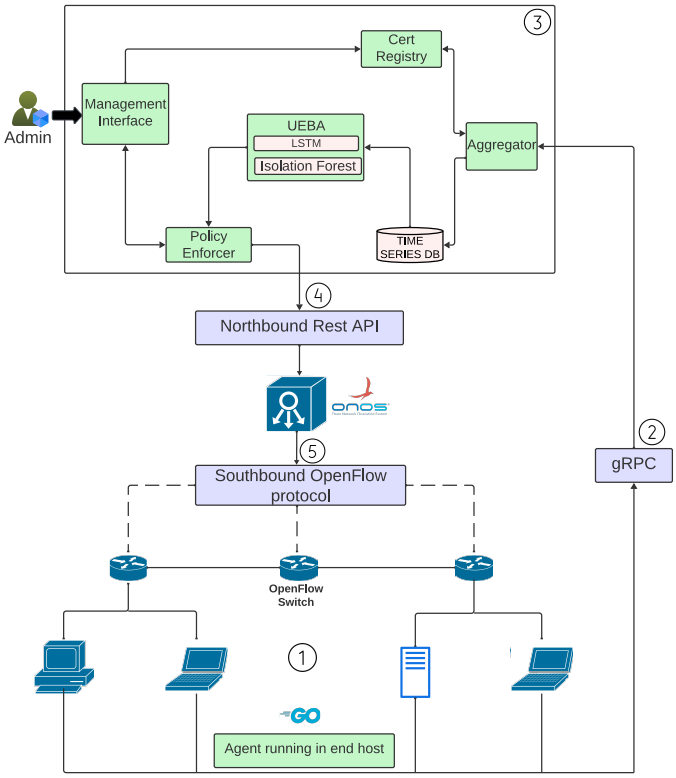

Fig. 1: High Level System Architecture

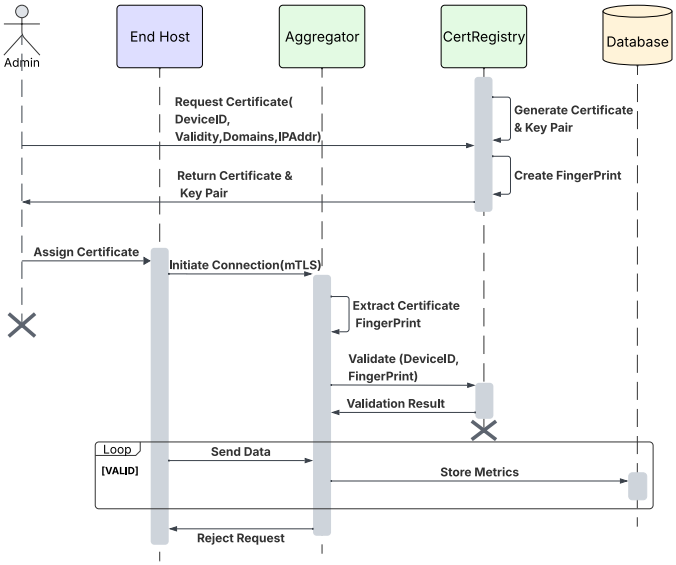

Fig. 2: Authentication Sequence Diagram

to model long-range dependencies and improve interpretability through generative or attention-based mechanisms [18]. While promising, many of these models are intended for batch processing and assume access to large, stable datasets. These assumptions make them poorly suited for real-time, SDN-integrated systems with limited computational resources.

Despite the strengths of the above approaches, most focus on network-layer traffic or rely on standalone detection pipelines disconnected from enforcement mechanisms. Our work differs by unifying host-level telemetry collection, machine-learning-based behavior modeling, and SDN-driven policy enforcement into a coherent and deployable system. By fusing security monitoring with dynamic network control, the proposed design addresses both detection and mitigation, thereby enabling proactive defense within SDN infrastructures.

III. System Design

This section details the system architecture of the proposed agent-based end-host monitoring system, as illustrated in Fig. 1. The system collects host-level telemetry, performs anomaly detection using a hybrid User and Entity Behavior Analytics (UEBA) framework, and enforces response policies through the SDN controller.

The pipeline begins at (1), where each monitored host runs a lightweight agent implemented in Go. These agents periodically collect system and process-level metrics, including CPU usage, memory, disk I/O, network activity, and per-process statistics. The schema used for data serialization is defined in Table I, and the collected metrics are serialized using Protocol Buffers[1] before being transmitted to the Aggregator via gRPC, as shown in (2) of Fig. 1. This connection uses mutual TLS (mTLS) to ensure message integrity, confidentiality, and peer authentication.

TABLE I: Monitoring Metrics Schema

Type	Field	Description (Units)
Metrics	hardware_metrics	System-wide metrics
	software_metrics	Process-specific metrics
HardwareMetrics	cpu_usage	CPU utilization (%)
	disk_usage	Storage usage (%)
	memory_usage	RAM usage (%)
	swap_usage	Swap usage (%)
	network	Network traffic stats
NetworkMetrics	packets_sent	Packets transmitted
	packets_recv	Packets received
SoftwareMetrics	name	Process identifier
	cpu_percent	CPU usage (%)
	mem_percent	Memory usage (%)
	pid	Process ID
	created_at	Creation timestamp
	threads	Thread count

The authentication mechanism is initiated during the agent's first connection attempt, as shown in Fig. 2. Each agent presents a signed X.509 certificate issued by the Cert Registry, which acts as a dedicated Certificate Authority within the system. The certificate is verified against the CA trust chain and host metadata is extracted. Upon successful verification, the session is accepted and a persistent secure channel is established using mTLS. This one-time authentication avoids repeated handshakes and ensures both confidentiality and authenticity for all subsequent telemetry transmission. Once this secure channel is established, the Aggregator validates the structure of incoming telemetry based on the predefined schema, and stores it in a time-series database such

[1]https://protobuf.dev/

as InfluxDB[2] or OpenTSDB[3]. The original timestamps are preserved during ingestion, enabling downstream analytics to evaluate both absolute and temporal behaviors.

The User and Entity Behavior Analytics (UEBA) service establishes behavioral baselines for entities by monitoring patterns in system metrics over time. These baselines capture temporal trends and are continuously monitored to identify deviations that signal security threats. The UEBA service asynchronously retrieves data from the time-series database, creates distinct baselines for each user, and analyzes live data against them to detect anomalous behavior. In our system, the UEBA service applies two machine learning models, namely Isolation Forest for system-level anomaly detection and an LSTM-based Autoencoder for modeling temporal patterns in process behavior. Their respective roles and mathematical formulations are described in the subsequent sections.

1) Isolation Forest: We employ Isolation Forest (iForest) [19], an unsupervised anomaly detection algorithm. It constructs an ensemble of binary trees, called *isolation trees* (iTrees), by recursively selecting a feature and splitting the dataset at a random threshold. Since anomalies are rare and distinct, they tend to be isolated in fewer splits, resulting in shorter path lengths in the tree structure.

The anomaly score of a data point x is computed based on the average path length across all trees in the forest:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{1}$$

where,

- $E(h(x))$ is the average path length of x in the isolation trees.
- $c(n)$ represents the average path length of unsuccessful searches in a binary search tree and is given by:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n} \tag{2}$$

where $H(n)$ is the harmonic number:

$$H(n) = \sum_{i=1}^{n} \frac{1}{i} \tag{3}$$

Anomaly scores close to 1 indicate potential outliers, whereas values near 0 suggest normal instances. In this work, iForest is employed for detecting hardware-level anomalies based on system metrics such as CPU usage, memory, and network packet activity. In our implementation, we utilize scikit-learn's Isolation Forest with 100 estimators.

2) LSTM-based Autoencoder: To detect anomalies in software behavior patterns, we adopt a Long Short-Term Memory (LSTM) [20] based autoencoder. This model is well-suited for learning temporal dependencies in sequential data, making it ideal for modeling software metric time series.

The encoding and decoding processes of autoencoder are defined as:

[2]https://www.influxdata.com/time-series-platform/influxdb/

[3]http://opentsdb.net

$$z = f_\theta(x) = \sigma(W_e x + b_e) \tag{1}$$

$$x' = g_\phi(z) = \sigma(W_d z + b_d) \tag{2}$$

where,

- $x \in \mathbb{R}^k$ is the input feature vector, with k dimensions.
- $z \in \mathbb{R}^d$ is the latent (compressed) representation, with $d \ll k$.
- $x' \in \mathbb{R}^k$ is the reconstructed approximation of the original input.
- $f_\theta(x)$ is the encoder function parameterized by $\theta = \{W_e, b_e\}$.
- $g_\phi(z)$ is the decoder function parameterized by $\phi = \{W_d, b_d\}$.
- $\sigma(\cdot)$ is an activation function

An LSTM autoencoder consists of two core components: an encoder that compresses input sequences into a latent representation, and a decoder that attempts to reconstruct the original input.

We implement this architecture in PyTorch with a sequence length of 10. Anomalies are detected based on the Mean Squared Error (MSE) between the original and reconstructed sequences.

Higher reconstruction errors indicate deviations from learned behavioral baselines, flagging potential anomalies. This technique efficiently captures subtle and complex temporal patterns in host-level process behavior that are often overlooked by traditional rule-based or statistical methods.

Once an anomaly is detected, the UEBA module reports it to the Policy Enforcer along with metadata such as the affected host, anomaly type, and confidence score. Administrators can define rules that govern how different types of anomalies are handled. Based on these rules, the system determines whether a response is warranted.

In (4), if enforcement is triggered, the Policy Enforcer communicates with the SDN controller via its northbound REST API. The controller then translates the received directives into flow rules, which are then installed on OpenFlow switches through its southbound interface (5). These flow rules can block, reroute, or isolate traffic associated with the affected host. This closes the loop from detection to enforcement, enabling near-real-time response to host-level anomalies within the SDN environment.

IV. Evaluation

All experiments were conducted on a workstation running Windows 11 Pro (Build 26100) with an Intel Core i7-13700KF CPU @ 3.20 GHz, 64 GB DDR5 RAM. The mTLS configuration used TLS 1.3 with 4096-bit RSA certificates in X.509 format, issued by a local certificate authority. Unless otherwise specified, all reported results are averaged over 5 independent runs.

A. Functional Validation

To demonstrate the operational correctness of the proposed system, we conducted a functional validation using a controlled deployment in a virtualized testbed. The experimental setup consisted of two virtual machines acting as monitored end hosts, each running our monitoring agent configured to collect system-level metrics at 5-second intervals. These agents streamed telemetry data to the Aggregator service over gRPC channels secured via mTLS. The Aggregator and UEBA services were hosted on a separate virtual machine, which also included an InfluxDB instance for metric storage. The Certificate Registry Service issued unique X.509 certificates to each agent, embedding host identifiers in the certificate's Common Name (CN) field for identity binding. An emulated ONOS northbound API interface was implemented to log policy decisions, substituting flow rule installation.

To validate anomaly detection and response mechanisms, we performed an anomaly injection experiment using the `stress-ng`[4] utility to induce CPU stress on one of the monitored hosts. The resource exhaustion event was initiated at $t = 00{:}22{:}01$.

The system's response unfolded as follows: a secure mTLS session was established with a handshake latency under 150 ms. Telemetry batches were ingested by the Aggregator within 200 ms per batch. After two sample intervals (10 seconds), the UEBA service flagged abnormal CPU usage exceeding 75%, triggering a policy update. This decision was relayed to the ONOS stub interface, which logged the action at $t = 00{:}22{:}11$, approximately 10 seconds after anomaly onset.

This validation confirms the correct end-to-end behavior of the monitoring pipeline, encompassing secure data transmission, anomaly detection, and reactive policy signaling as demonstrated by the controlled prototype evaluation.

B. Performance and Resource Overhead

1) Agent Footprint: The monitoring agent exhibited a consistent CPU load of approximately 2.0% per core, with short-term spikes up to 2.5% during metric collection intervals. Average heap memory usage remained stable at 7.72 MB throughout normal operation. Disk I/O activity was minimal, limited primarily to local logging. Network traffic remained low, with each agent transmitting approximately 2.01 Kbps to the aggregator and receiving 0.47 Kbps in return.

2) mTLS Connection Overhead: To quantify the cost of establishing secure channels, we conducted benchmark tests involving 10,000 consecutive mTLS handshakes. The mean handshake time was 7.86 ms ($\sigma = 0.53$ ms), with observed bounds between 6.10 ms and 12.22 ms. These measurements were performed over the loopback interface; real-world deployments across physical links would incur additional network latency. The impact of this initial handshake and certificate validation overhead is effectively managed in our implementation by employing gRPC streaming with certificate validation performed only once at connection initialization. The system extracts the agent's identity from the Common Name (CN) field of its X.509 certificate during this handshake and maintains the authenticated context for the duration of the stream. This design eliminates the need for repeated validation across individual RPCs, reducing cryptographic overhead and improving efficiency for long-lived telemetry sessions.

[4]https://github.com/ColinIanKing/stress-ng

C. Scalability Testing

1) Aggregator Load Performance: To evaluate how well our aggregator scales under load, we simulated increasing numbers of telemetry agents from 1 to 1000, each maintaining a persistent mTLS connection and streaming synthetic telemetry data to the aggregator. Table II summarizes CPU utilization, memory consumption, network throughput, and stream processing latency under varying host counts.

The aggregator remains efficient under increasing load: CPU and memory usage scale sub-linearly, and network throughput grows predictably and linearly. However, beyond 500 hosts, we observe a significant increase in mean latency and its variance. These increases are likely attributable to gRPC I/O contention or Go runtime scheduling overhead. Beyond 750 hosts, we observe pronounced latency outliers, likely due to garbage collection pauses or other runtime-induced stalls, signaling stress points for the system. These findings validate our design for small to medium-sized enterprise deployments while identifying targets for future I/O profiling and tuning.

2) UEBA Load Performance: To assess the scalability of the UEBA component, we measured its resource usage while processing telemetry streams from increasing numbers of simulated hosts. Table III summarizes CPU and memory utilization. CPU usage grew sub-linearly up to 500 hosts and then plateaued, suggesting compute saturation. Memory usage scaled linearly, reflecting per-host state growth. The UEBA system remained responsive up to 1000 hosts, demonstrating viability for moderate-scale deployments.

D. Anomaly Detection Accuracy

To evaluate the anomaly detection capabilities of the proposed framework, we conducted controlled anomaly injection experiments. Prior to anomaly injection, a behavioral baseline was established using 4,587 instances of normal system operation. Six types of anomalies—namely CPU exhaustion, memory saturation, disk I/O stress, socket flooding, swap overuse, and excessive network usage were induced using the `stress-ng` utility. Additionally, process-level anomalies were introduced by executing previously unseen applications on monitored hosts.

Each anomaly was injected for a fixed duration of 60 seconds, and system metrics were collected at 5-second intervals for a total monitoring duration of approximately 8 minutes per experiment. Based on observations, anomalies in system-level metrics typically manifested starting from the second interval after injection (i.e., at time $t+2$), whereas deviations in software-level behavior were detected slightly later—typically around $t+4$ to $t+5$, due to the use of sequence-based modeling in the software anomaly detection pipeline.

TABLE II: Aggregator Scalability under Varying Host Counts

Host Count	CPU (%)	Memory (MB)	Net RX (KB/s)	Net TX (KB/s)	Latency (ms)
0 (Idle)	0.00 ± 0.00	5.17 ± 0.04	–	–	–
1	0.01 ± 0.03	5.90 ± 0.57	0.27	0.67	0.26 ± 0.70
2	0.03 ± 0.08	8.88 ± 1.41	0.57	1.42	0.21 ± 0.71
5	0.09 ± 0.19	11.03 ± 0.28	1.49	3.47	0.73 ± 0.80
10	0.55 ± 1.10	14.18 ± 0.42	3.05	7.30	0.63 ± 0.72
50	1.11 ± 1.17	26.17 ± 0.24	13.96	33.92	5.00 ± 3.94
100	2.25 ± 0.96	43.92 ± 0.29	28.56	69.67	15.13 ± 8.03
250	4.58 ± 2.00	60.75 ± 0.35	72.92	175.64	30.21 ± 15.04
500	7.51 ± 2.72	80.81 ± 0.38	142.94	345.36	71.04 ± 29.97
750	6.26 ± 7.83	107.93 ± 23.23	278.76	492.65	160.05 ± 60.04
1000	5.71 ± 11.80	162.25 ± 14.81	428.69	598.52	258.86 ± 74.83

TABLE III: UEBA Scalability under Varying Host Counts

Hosts Count	CPU (%)	Memory (MB)
0	0.25 ± 1.67	166.89 ± 16.82
1	0.47 ± 2.44	605.72 ± 3.96
2	0.51 ± 0.54	609.26 ± 5.36
5	0.58 ± 0.32	617.06 ± 6.85
10	0.84 ± 2.24	644.22 ± 15.27
50	2.44 ± 3.79	780.91 ± 20.22
100	4.85 ± 4.38	900.09 ± 21.29
250	9.22 ± 4.86	1148.77 ± 110.37
500	16.01 ± 5.52	1484.27 ± 278.74
750	17.44 ± 6.30	1800.03 ± 379.22
1000	18.18 ± 5.57	2192.46 ± 618.08

Detection performance was quantified using standard evaluation metrics—precision, recall, F1-score, and accuracy as defined below:

- Precision = $\frac{TP}{TP+FP}$
- Recall = $\frac{TP}{TP+FN}$
- F1-Score = $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$
- Accuracy = $\frac{TP+TN}{TP+FP+TN+FN}$

The Isolation Forest model was used to detect system-level anomalies based on hardware metrics, while a sequence-aware LSTM Autoencoder was employed for identifying anomalous behavior in process-level features. Both models effectively detected injected anomalies in their respective domains.

TABLE IV: Anomaly Detection Evaluation Metrics

Metric Type	Precision	Recall	F1-Score	Accuracy
System-Level	0.60	0.84	0.70	0.92
Software-Level	0.85	0.75	0.799	0.96

V. Discussion

A. Scalability Observations and Design Viability

Our scalability evaluation (Sec. IV), shows that the aggregator maintains low and predictable CPU and memory usage up to 500 concurrent hosts, with latency remaining within acceptable bounds. Beyond this threshold, latency increases significantly, indicating a bottleneck likely stemming from gRPC I/O contention and runtime scheduling limitations, including garbage collection and goroutine scheduling overhead. The UEBA service remained operational and responsive up to 1000 hosts, though CPU utilization approached saturation beyond 500 hosts, suggesting compute resource limits. Memory usage scaled linearly, reflecting per-host state maintenance. These results affirm the architectural viability of the system for small to medium-scale deployments. Scaling beyond this range will require architectural modifications, including sharded aggregators or service-mesh-based telemetry pipelines, as well as distributed or stateless UEBA microservices to parallelize inference across hosts.

B. Deployment Considerations for SDN Intranets

The system integrates host-level anomaly detection with SDN control to enable dynamic, policy-driven network defense. Evaluation showed that resource anomalies were typically detected and signaled within 10 seconds of onset, demonstrating the feasibility of near real-time response in a controlled setting. However, automated enforcement in production requires careful safeguards. False positives could lead to unintended isolation or traffic disruption. Future deployments must incorporate techniques such as confidence-aware thresholds, corroboration across multiple signals, and administrative validation steps to ensure robust and safe operation.

C. UEBA Module Role and Limitations

The UEBA module served as a prototype to validate the feasibility of host-behavior-based detection within an SDN-secured intranet. In this work, it focused on system and process-level metrics. For real-world deployments, the detection pipeline can be enhanced by integrating richer data sources such as system logs, user session patterns, authentication events, and system call traces. Additionally, future systems should explore online learning, concept drift detection, adversarial robustness, and model distillation to support low-latency, adaptive analytics at the edge.

D. Trade-offs of Simulation vs. Real Testbed

Our evaluation was conducted in a containerized virtual testbed, which enabled controlled experimentation, anomaly injection, and repeatable results. However, such environments abstract away important real-world factors including switch processing delays, flow setup time, and heterogeneous traffic patterns. Validation on physical SDN infrastructure is necessary to assess latency, detection accuracy, and control loop stability under realistic conditions.

E. Implications for Deployment in Production SDN Networks

Integrating host-based anomaly detection into SDN-controlled environments offers significant promise but also presents challenges. Dynamic flow rule installation triggered by real-time host anomalies must balance responsiveness with stability to avoid unintended disruptions. The ability to issue fine-grained, per-host isolation policies directly from the UEBA system via SDN northbound APIs demonstrates strong potential for enhancing network resilience. However, production deployments would require stricter safeguards, such as verification loops or multi-factor anomaly confirmation, to prevent false-positive triggered isolations.

VI. Conclusion and Future Work

This paper presented an agent-based end-host monitoring system that integrates secure telemetry collection and UEBA-driven anomaly detection with dynamic, per-host policy enforcement via SDN controller APIs. The proposed architecture demonstrates the feasibility of practical near-real-time anomaly detection in SDN-managed intranets. Evaluation results confirmed the system's ability to detect both hardware and software-level anomalies with low per-agent overhead, and demonstrated its viability for small to medium-scale deployments, with measured latency and resource utilization within acceptable operational thresholds. The system remains a prototype evaluated in simulation, requiring real-world validation and architectural refinement for deployment readiness.

Future work should address these gaps by: (i) implementing sharded aggregators and horizontally scalable UEBA microservices using service mesh or container orchestration; (ii) deploying and evaluating the system in a physical SDN testbed to assess latency, resilience, and control-plane behavior under real-world conditions; (iii) developing lightweight, edge-deployable models, such as compressed, distilled LSTM autoencoders, to enable localized anomaly detection without centralized dependence; (iv) integrating additional behavioral signals such as system logs, syscall traces, and user session patterns to enrich anomaly context and reduce false positives; (v) conducting comparative evaluation against existing host-based SDN anomaly detection frameworks to benchmark detection accuracy, latency, and system overhead under consistent testing conditions.

Collectively, these enhancements can advance the system from a working prototype to a deployable framework for securing next-generation SDN-managed intranets.

VII. Acknowledgments

This work was partly supported partly by Japan Society for the Promotion of Science (JSPS) KAKENHI Grant Number 23K05416 (Grant-in-Aid for Scientific Research(C)) and University Grants Commission, Nepal (Grants ID: CRG-078/79-ENgg-01).

Appendix

The source code for the components discussed in this paper is available in the following repositories:

Component	Repository URL
Sherlock (Agent)	https://github.com/Camph0r/sherlock
Watson (Monitoring)	https://github.com/Camph0r/watson

References

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

[2] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in software defined networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2317–2346, 2015.

[3] L. Csikor and D. P. Pezaros, "End-host driven troubleshooting architecture for software-defined networking," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pp. 1–7, 2017.

[4] M. E. Najd and C. A. Shue, "Deepcontext: An openflow-compatible, host-based sdn for enterprise networks," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pp. 112–119, 2017.

[5] T. Choi, S. Song, H. Park, S. Yoon, and S. Yang, "Suma: Software-defined unified monitoring agent for sdn," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–5, 2014.

[6] C. Katsis and E. Bertino, "Zt-sdn: An ml-powered zero-trust architecture for software-defined networks," *ACM Trans. Priv. Secur.*, vol. 28, Feb. 2025.

[7] G. Yang, Y. Yoo, M. Kang, H. Jin, and C. Yoo, "Accurate and efficient monitoring for virtualized sdn in clouds," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 229–246, 2023.

[8] A. G. Martín, A. Fernández-Isabel, I. Martín de Diego, and M. Beltrán, "A survey for user behavior analysis based on machine learning techniques: current models and applications," *Applied Intelligence*, vol. 51, no. 8, pp. 6029–6055, 2021.

[9] M. Shashanka, M.-Y. Shen, and J. Wang, "User and entity behavior analytics for enterprise security," in *2016 IEEE International Conference on Big Data (Big Data)*, pp. 1867–1874, 2016.

[10] S. Iglesias Perez and R. Criado, "Increasing the effectiveness of network intrusion detection systems (nidss) by using multiplex networks and visibility graphs," *Mathematics*, vol. 11, no. 1, p. 107, 2022.

[11] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau, "Autoencoder-based network anomaly detection," in *2018 Wireless Telecommunications Symposium (WTS)*, IEEE, 2018.

[12] S. Maleki, S. Maleki, and N. R. Jennings, "Unsupervised anomaly detection with lstm autoencoders using statistical data-filtering," *Applied Soft Computing*, vol. 108, p. 107443, 2021.

[13] M. S. Elsayed, N.-A. Le-Khac, S. Dev, and A. D. Jurcut, "Network anomaly detection using lstm based autoencoder," in *Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet '20)*, pp. 37–45, ACM, 2020.

[14] A. Vikram and Mohana, "Anomaly detection in network traffic using unsupervised machine learning approach," in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, (Coimbatore, India), IEEE, 2020.

[15] X. Chun-Hui, S. Chen, B. Cong-Xiao, and L. Xing, "Anomaly detection in network management system based on isolation forest," in *2018 4th Annual International Conference on Network and Information Systems for Computers (ICNISC)*, (Wuhan, China), IEEE, 2018.

[16] X. Chen, L. Deng, F. Huang, C. Zhang, Z. Zhang, Y. Zhao, and K. Zheng, "Daemon: Unsupervised anomaly detection and interpretation for multivariate time series," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 2225–2230, 2021.

[17] Y. Li, X. Peng, J. Zhang, Z. Li, and M. Wen, "Dct-gan: Dilated convolutional transformer-based gan for time series anomaly detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3632–3644, 2023.

[18] Z. Zamanzadeh Darban, G. I. Webb, S. Pan, C. Aggarwal, and M. Salehi, "Deep learning for time series anomaly detection: A survey," *ACM Comput. Surv.*, vol. 57, Oct. 2024.

[19] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*, pp. 413–422, IEEE, 2008.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.