International Journal of Safety and Security Engineering
Vol. 16, No. 6, June, 2026, pp. 1333-1342
Journal homepage: http://iieta.org/journals/ijsse

FL-NDR: A Federated Unlearning-Driven Network Detection and Response System for DDoS Defence in Software-Defined Networks

Noble T N Pillai*, Meena Mathivanan

Department of Electronics and Communication Engineering, Vels Institute of Science, Technology & Advanced Studies (VISTAS), Chennai 600 117, India

Corresponding Author Email: nobletnpillai@gmail.com

https://doi.org/10.18280/ijsse.160615

Received: 10 May 2026
Revised: 11 June 2026
Accepted: 22 June 2026
Available online: 30 June 2026

Keywords:
federated unlearning, Software Defined Networking, Distributed Denial of Service detection, model poisoning, intrusion detection and response, OpenFlow telemetry, Privacy-Preserving machine learning, label-flip attack

ABSTRACT

Software Defined Networking (SDN) gives us a programmable network where a controller makes all the forwarding decisions. This is powerful for automation, but it also means one point in the network becomes a target. If someone takes the controller offline with a Distributed Denial of Service (DDoS) attack, the entire network could go down. Machine learning classifiers can detect these attacks from edge devices, but training a centralised model means sharing raw traffic from all domains-which often leads to capacity and privacy problems. Federated Learning (FL) technique solves this issue: each domain trains locally and sends only model updates to a server, which combines them into a global model. No raw traffic leaves any domain. The problem is that a compromised domain can send manipulated updates to quietly reduce the model's ability to detect attacks. Even after we discover the bad client and remove it, its past updates are already mixed into the model. This is called model poisoning, and federated unlearning is the process of removing that influence. This paper introduces Federated Unlearning Network Detection and Response (FL-NDR). It treats every OpenFlow switch in the network as a sensor. Together, all the switches in the fabric form a distributed sensor layer that feeds data into a federated DDoS detection system. When a poisoned client is found, FL-NDR does two things at once: it rolls back the global model to a clean checkpoint, and it tells the SDN controller to block the compromised domain using OpenFlow flow rules. We validate this through an SDN simulation and through six experimental scenarios (S1-S6) on the Mendeley SDN DDoS dataset. The best configuration-rapid retraining from a pre-poisoning checkpoint with client quarantine (S6)-recovers detection accuracy to 98.81%, matching the clean baseline. The results show that network-level isolation and model-level unlearning must be used together. Neither works on its own.

1. INTRODUCTION

Software Defined Networking (SDN) separates the control plane from the data plane. The controller decides how traffic is forwarded, and the switches follow those instructions. This makes the network easy to manage and automate. The problem is that all forwarding intelligence lives in one place. A Distributed Denial of Service (DDoS) attack against the controller can quickly flood its flow tables, saturate the OpenFlow channel, and exhaust its CPU at the same time. When the controller goes down, the entire network stops working.

The obvious fix is to use machine learning to detect DDoS attacks from switch statistics before they cause damage. Most approaches train a single classifier on traffic collected from all network domains. This works well technically, but it means every domain has to share its raw traffic with a central server. In practice, organisations are not willing to do this. There are privacy concerns, a lack of required capacity, and, in some industries, regulatory constraints.

Federated Learning (FL) was designed to solve exactly this problem. Each domain trains a local model on its own traffic. Only the model weights-not the raw traffic-are sent to a central server. The server combines all the updates using FedAvg and sends an improved global model back to all clients. This way every domain contributes to a shared model without sharing sensitive data or requiring additional network infrastructure capacity.

There is still one problem. A compromised domain can send manipulated model updates to the server. The server cannot distinguish these from legitimate updates. Over time, the poisoned updates reduce the global model's ability to detect DDoS attacks. This is called model poisoning. Even if we discover the bad client and stop accepting its updates, the damage is already done. Its past contributions are mixed into the model weights. Simply removing the client from future rounds does not fix the model. This is where federated unlearning comes in. Unlearning actively removes the influence of a specified client from the already-trained model.

This paper goes a step further. Federated Unlearning Network Detection and Responses (FL-NDR) treats the SDN network itself as a distributed sensor layer. Every OpenFlow

switch already collects per-flow statistics as part of normal operation. FL-NDR uses this telemetry as the data source for both detection and the unlearning pipeline. When a poisoned client is found, FL-NDR does two things at once: it rolls back the global model to a checkpoint from before the poisoning started, and it installs OpenFlow rules at the compromised domain's border switches to block all its inter-domain traffic. Both the model and the network are fixed at the same time.

The main contributions of this paper are:

- We show that every OpenFlow switch can act as a network sensor using only the built-in Statistics API, with no additional hardware.
- We design FL-NDR, a four-layer NDR system that combines distributed sensing, federated detection, federated unlearning, and programmatic OpenFlow response.
- We validate the FL-NDR through two distinct evaluations: an SDN simulation for the sensor fabric and response layer, and a six-scenario federated unlearning experiment (S1 to S6) on the Mendeley SDN DDoS dataset [1]. The best result (rapid retraining with quarantine) recovers accuracy to 98.81%.

The remaining part of the article is organised as follows: Section 2 describes existing methods and previous works that contribute to improving security in SDN; Section 3 describes the proposed work; Section 4 describes experimental analysis; and Section 5 concludes our work.

2. RELATED WORK

SDN and FL have both attracted significant security research in recent years. This section covers four areas that this work builds on: DDoS detection in SDN, Network Detection and Response (NDR), FL for security, and adversarial attacks with federated unlearning.

2.1 Distributed Denial of Service attack detection in Software Defined Networking

The DDoS attack detection in SDN has been studied from many angles. One recurring challenge is handling the volume and variety of attack traffic without overwhelming the controller. Yu et al. [2] point out that any detection framework also needs to keep communication and storage overhead low, particularly in resource-constrained environments like satellite-to-device networks.

Various hybrid models have delivered strong performance results for multi-class detection tasks. Deep Neural Networks (DNN) [3] and Long Short Term Memory (LSTM) in parallel, processing complex features and sequential patterns separately before merging outputs. This achieved 98.84% accuracy. To make the DDoS attack detection work in high-volume SDN environments, PCA [4], a technique to reduce dimensionality before feeding data into a 1D Convolutional Neral Network (CNN) for real-time mitigation.

Not all attacks arrive at the same rate. Adaptive two-stage DDoS [5] attack detection scheme with dynamic threshold adjustment (ATS-DTA), a two-stage scheme that uses conditional entropy to decide when to activate a full ML detection module. This saves controller resources during normal traffic. Bahashwan et al. [6] took a different approach to multi-rate threats: they combined Chi-Square and Random Forest feature selection to identify the most useful features for both high-rate and low-rate flooding attacks.

Network topologies keep evolving, Dynamic Graph Neural Network [7] that uses gated convolutional temporal layers and port grouping to track changes in the network structure. Naeen et al. [8] focused on imbalanced datasets and introduced an Attention-Enhanced Cross-Entropy model with a hierarchical attention mechanism. For low-level threats, a kernel-level [9] eBPF/XDP framework using Exponential Moving Averages for dynamic thresholding. Wang and Wang [10] combined time-frequency analysis with a spatial-temporal graph network to catch low-rate attacks.

Drawback: Most of these systems depend on datasets that are not SDN-specific, or they require dedicated monitoring hardware that adds deployment cost and resource contention.

Contribution: FL-NDR removes this hardware requirement. Every OpenFlow switch already exposes flow statistics through the built-in Statistics API. FL-NDR uses these directly, with no extra sensors needed.

2.2 Network detection and response

NDR systems are designed to bridge the gap between detecting the adversarial activities and responding to them by enabling automated mitigation mechanisms. A combined SDN with blockchain-based routing for scalable authentication in AIoT networks [11]. Macas et al. [12] reviewed how vulnerable deep learning models are to adversarial examples. They showed that even models with strong performance can be deceived by carefully crafted inputs.

DeepSDN [13] as a dedicated security approach for SDN environments. Their method relies on (a) adaptive threshold scoring, (b) allowing the detection logic to adjust dynamically and thereby improve real-time threat detection accuracy. In addition, it incorporates blockchain to support secure and reliable transaction verification, helping preserve the integrity and trustworthiness of recorded network activities.

To address Advanced Persistent Threats (APT) Privacy-Preserving Provenance Graph-Based Model for Autonomous APT Detection [14] in SDN (P3GNN). P3GNN is a provenance-graph-based model that captures and analyzes activity relationships over time. It traces malicious behavior at the node level within an SDN framework, helping identify how an attack progresses inside the network.

Drawback: Current NDR systems detect attacks and mitigate them, but they do not handle the case where a FL participant has been poisoning the detection model itself. There is no mechanism to remove that malicious influence from the global model.

Contribution: FL-NDR closes this gap. It provides a complete pipeline: distributed sensing, federated detection, programmatic OpenFlow response, and federated unlearning-all in one coordinated system.

2.3 Federated Learning for network security

The application of FL as a tool for network security enhancement has become very useful because it allows multiple domains to collaborate without sharing raw traffic. Fed-Adapt [15], which uses Byzantine-resilient FL to reconfigure SDN topology during volumetric attacks in under a second.

Fed-Evolver [16] as a federated intrusion detection approach designed for environments where labeled training data is scarce or difficult to obtain. The method uses

adversarial autoencoders in a semi-supervised learning setup, so it can learn useful representations from largely unlabeled traffic while still benefiting from the limited labeled samples that are available. This design helps the global model remain stable and accurate even as network traffic patterns evolve over time (for example, due to workload changes, new applications, or shifting attacker behaviors).

Yasarathna and Le-Khac [17] reviewed adversarial threats to deep learning-based detection systems and found that accuracy can drop by up to 48.40% under targeted perturbations. This finding confirms that even a well-designed FL system need a defence against poisoning.

Drawback: Existing FL security frameworks do not have a built-in recovery mechanism. When a poisoning attack is discovered, the usual fix is to retrain from scratch. This is expensive and not practical for real-time systems.

Contribution: The FL-NDR validates two retraining strategies, (1) the scenario S4 (rapid, no quarantine) and (2) the scenario S6 (rapid with quarantine). S6 recovers model accuracy to 98.70% significantly faster than full retraining, while keeping the compromised client isolated throughout recovery process.

2.4 Adversarial attacks and unlearning

Several recent works around AI security have addressed how to remove malicious influence from a trained model. cost-aware federated unlearning framework [18] that uses a sliding window to limit how much of the model needs to be updated, reducing computational cost. A combined knowledge distillation [19] with unlearning to defend against gradient inversion attacks. Lu et al. [20] examined an approach with feature-level unlearning within the scope of vertical FL. Their work aims to erase sensitive feature representations held by active clients, so that private information is no longer encoded in the learned model. FedASU [21] for graph-based FL. It uses attention-guided sensitivity to identify which parts of the model are most influenced by the data targeted for removal, and then updates only those affected parameters to perform more focused unlearning. Diffusive noise injection [22] provides a fast approach to removing data from a model without requiring full retraining. Wang et al. [23] provided a broader review of poisoning attacks and the defenses against them, including cancellation methods where unlearning is used to undo the impact of malicious triggers.

Drawback: A lot of unlearning techniques end up lowering accuracy for clients that were never attacked, and this is especially noticeable in complex, graph-based setups. They also tend to be computationally expensive, which makes them harder to use in practice.

Contribution: In Scenario S6, we use the rapid retraining with a quarantine step to bring accuracy to 98.81% after a poisoning attack. The compromised client is kept separate during recovery, so the honest clients are not impacted.

3. FEDERATED UNLEARNING NETWORK DETECTION AND RESPONSE SYSTEM DESIGN

3.1 The Software Defined Networking fabric as a distributed sensor layer

Every OpenFlow switch maintains a flow table that records statistics for every active flow: packet counts, byte counts, flow duration, and queue depth. The controller can query these at any time using three OpenFlow primitives: OFPMP_PORT_STATS for per-port counters, OFPMP_FLOW_STATS for per-flow counters, and OFPMP_AGGREGATE_STATS for switch-level aggregates. FL-NDR treats every switch as a sensor. Together, all switches in the network form a dense sensor layer. Coverage is automatic: wherever traffic flows, it is measured.

Each FL domain deploys a Sensor Controller Agent (SCA). The SCA polls all switches in the domain and assembles feature vectors. There are 14 features organised across three levels. L1 captures port-level data: packet rate, byte rate, and Rx/Tx ratios. L2 captures flow-level data: duration, packet count, source IP entropy, destination port entropy, short-flow ratio, high-packet-count flow ratio, and TCP ratio. L3 captures switch-level aggregates: total flow count, Packet-In rate, flow-table utilisation, and table miss rate.

Source IP entropy is the most useful feature for detecting DDoS threats. During a spoofed-source flood, hundreds of unique IP addresses appear in the flow table at once. This pushes entropy up sharply. At the same time, Packet-In rate spikes because the switch sees flows it has no rules for and asks the controller what to do. Table utilisation also rises as the TCAM fills with short-lived attack entries. These three indicators together are a clear DDoS signature.

3.2 Four layer architecture

Traditional network monitoring uses dedicated hardware-network taps, flow exporters, or port mirrors-placed at specific points in the network. This is expensive. There are always gaps in coverage between sensor locations.

SDN fabric solves this problem natively without any need for additional hardware components or architectural modifications.

FL-NDR has four layers. Each layer has a specific job. Table 1 summarises them and Figures 1 and 2 illustrate the design.

Compared with the undefended baseline, the extra cost is limited, small and is only 16.2% overhead.

Table 1. Federated Unlearning Network Detection and Response (FL-NDR) four-layer architecture

Layer	What It Does	Software Defined Networking (SDN) Component
L1: Distributed Sensor	Collects switch telemetry and builds feature vectors	OpenFlow switches + Statistics API
L2: Federated Detection	Trains an MLP classifier via FedAvg and runs inference	Federated Learning (FL) clients + aggregation server
L3: Unlearning Pipeline	Detects poisoning, rolls back the model, manages quarantine	Aggregation server + checkpoint store
L4: Response Orchestration	Converts detection events into OpenFlow flow rules	SDN controller + southbound API

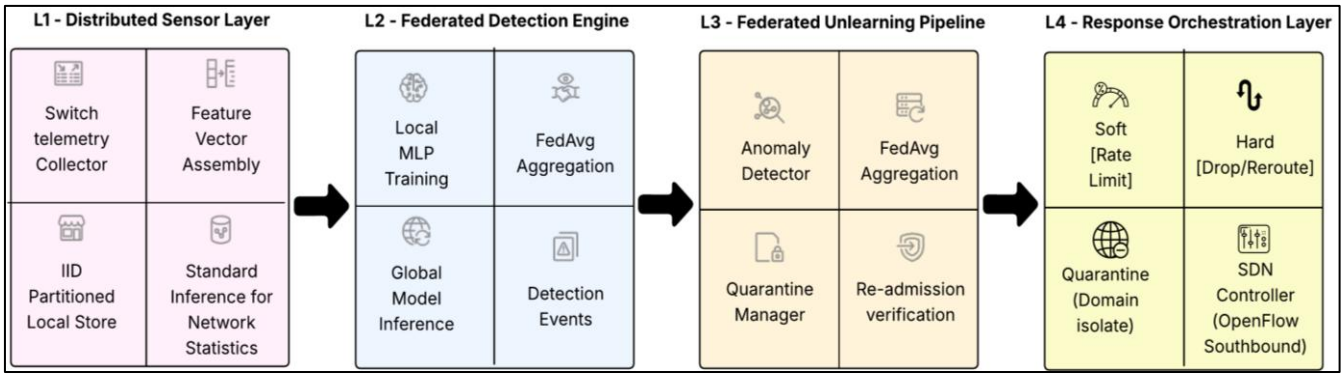

Figure 1. Federated Unlearning Network Detection and Response (FL-NDR) four-layer architecture: telemetry (L1), detection (L2), unlearning (L3), and response orchestration (L4)

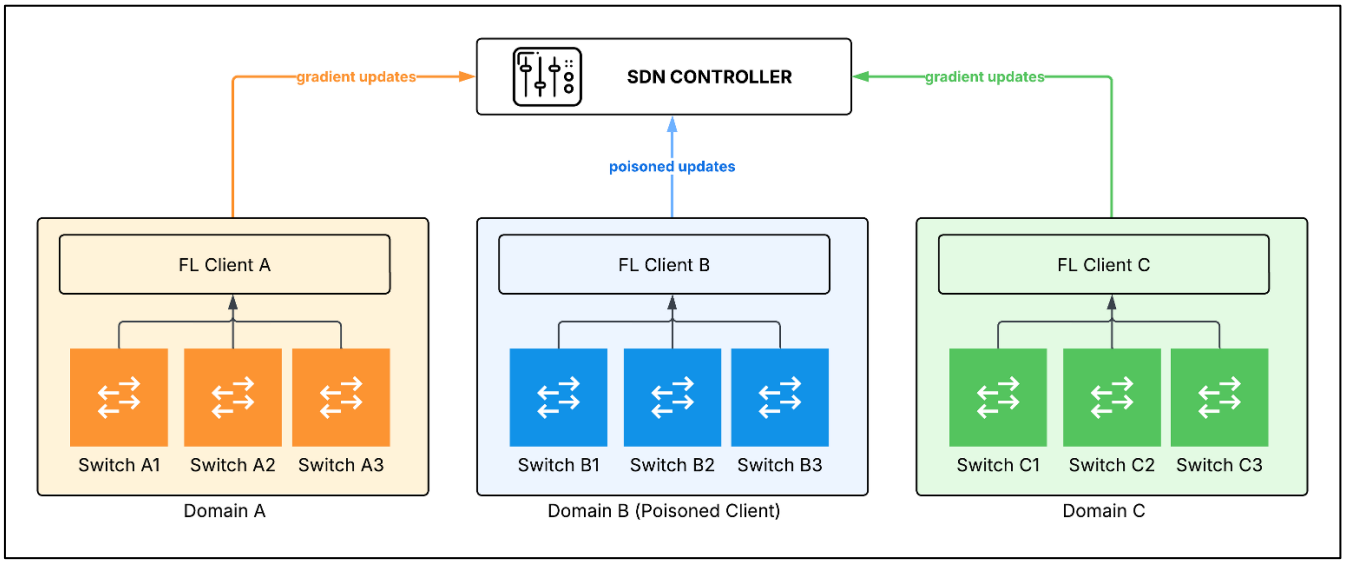

Figure 2. Software Defined Networking (SDN) sensor fabric with OFPMP telemetry collection and federated gradient aggregation

3.3 Response Orchestration Layer

The Response Orchestration Layer (ROL) sits at L4. It receives detection events from the federated detection engine and converts them into OpenFlow flow rules. There are three response tiers, chosen by confidence level.

T1 is a level-one soft response in the proposed approach. It fires when detection confidence is above 0.70. The ROL rate-limits suspect flows using OFPAT_SET_QUEUE and mirrors them to an analysis port. All services continue to run during this period.

Following next is T2, which is the hard response. It fires when confidence is above 0.90. The ROL drops flows matching the attack signature using OFPAT_DROP and reroutes legitimate traffic through a backup path using OFPIT_GOTO_TABLE.

T3 is the quarantine response. It fires when the Federated Unlearning Pipeline (FUP) confirms that an FL client is compromised. The ROL installs OFPIT_CLEAR_ACTIONS on all border switches of the suspect domain. This blocks all inter-domain traffic from that domain until quarantine is lifted.

All three tiers use the same OpenFlow channel as normal forwarding. There is no separate integration to build. A detection event can become an active flow rule in sub-second time.

3.4 Federated Unlearning Pipeline

The FUP operates at L3. The FL framework used is Plato [24, 25], a software platform designed to support scalable, reproducible, and extensible FL research. The framework's unlearning capabilities are seamlessly integrated into FL-NDR.

When it detects a compromised client, it initiates three actions in parallel as follows: (1) it blocks further gradient updates from that client, (2) rolls the global model back to the most recent clean checkpoint and resumes recovery training without the client, and (3) triggers the ROL to execute a T3 quarantine action.

The key point of interest is that all three happen at the same time. If we fix the model but leave the network un-isolated, the compromised domain can still relay attack traffic while the model is recovering. If we isolate the network but leave the model poisoned, detection stays broken. Hence, both must happen at the same time.

Similar to the above approach, there are two retraining strategies. Full retraining resets the global model to the round-0 checkpoint which is in effect, the randomly initialised weights before any training happened. This guarantees no poisoned weights remain, but it also throws away everything the honest clients learned. Rapid retraining restores the checkpoint from just before the compromised client first

participated. This preserves the honest clients' accumulated knowledge and leads to faster recovery.

In the quarantine variants, the compromised client is excluded from all recovery rounds. It cannot be re-admitted until two conditions are met-(1) the global model must reach 98% of baseline accuracy, (2) and the client's local model must have a cosine similarity of at least 0.95 with the global model.

4. SOFTWARE DEFINED NETWORKING SIMULATION VALIDATION

Before running the FL experiments, we validate the sensor fabric and the ROL using a Python-based SDN simulation. The simulation does not require Mininet. It also does not need real network hardware. It creates 50 OpenFlow switches across 10 domains. Each domain has 5 switches. The setup includes one SDN controller.

Each switch generates realistic flow-table entries. Also, each one of them exposes the same OpenFlow primitives as real hardware. These include but are not limited to OFPMP_PORT_STATS, OFPMP_FLOW_STATS, and OFPMP_AGGREGATE_STATS. Domain 2 receives a DDoS injection at round 5. The T3 quarantine triggers in the same round. This matches the unlearning trigger in the FL notebook.

4.1 Sensor telemetry signatures

Figure 3 shows six features from Domain 2's switches across 20 simulation rounds. Rounds 1 to 4 are clean. All features are stable. At round 5 the attack starts. Source IP entropy jumps from 3.28 to 5.17. Packet-In rate jumps from 23.8 to above 160. Short-flow ratio jumps from 0 to 0.85. Table utilisation double. These four features all move at the moment of attack. This confirms that the three-level OFPMP telemetry produces a clear and detectable DDoS signature without any additional hardware components or the underlying infrastructure modifications.

From round 5 to round 13, all features drop to zero. This is the expected result of T3 quarantine. The ROL installs OFPIT_CLEAR_ACTIONS at Domain 2's border switches and blocks all traffic. The switch no longer receives flows, so all statistics go to zero. This is not a detection failure-it is the quarantine working correctly. At round 14, quarantine is lifted and Domain 2 is re-admitted. The attack continues, and the signatures immediately reappear. The sensor layer is still working.

4.2 Response Orchestration Layer behaviour

Figure 4 shows when each ROL tier fired across the 20 simulation rounds. T3 fires at round 5. This is the red diamond at the top of the chart. At the same moment, the FUP triggers the model rollback. Both happen in the same round. At round 14, quarantine is lifted. This is the blue diamond. From round 15 onwards, the attack traffic resumes and the model successfully detects it with high confidence. T2 hard-drop rules fire in every round from 15 to 20. These are the orange dots in the middle row. T1 never fires. The pre-trained classifier classifies attack flows above the T2 threshold from the first detection, so the soft response is never needed. This proportional escalation-T3 for federation compromise, T2 for confirmed attack traffic -matches the three-tier design from Section 3.3.

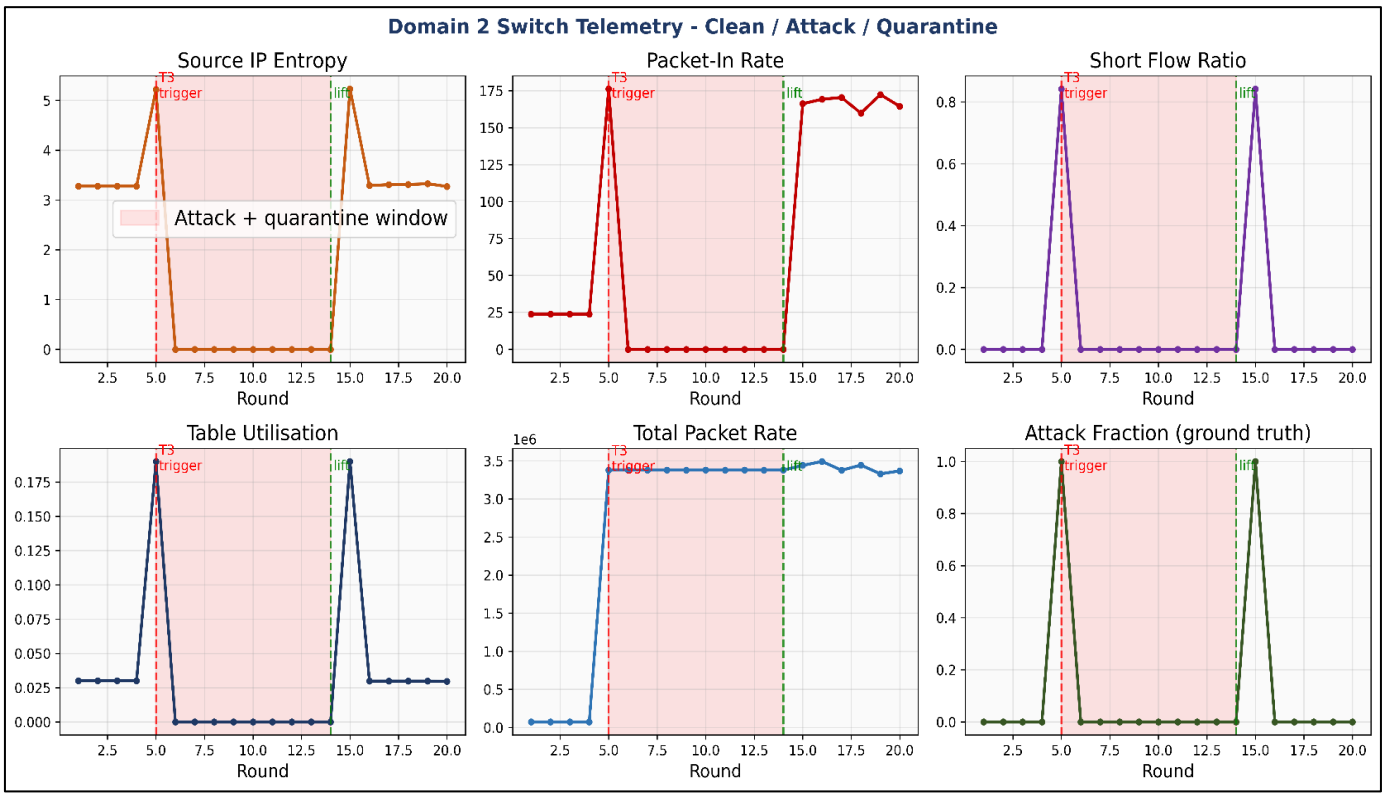

Figure 3. Domain 2 telemetry over 20 rounds. Features spike at attack onset (round 5), drop to zero during T3 quarantine (rounds 5–13), and reappear after quarantine is lifted (round 14)

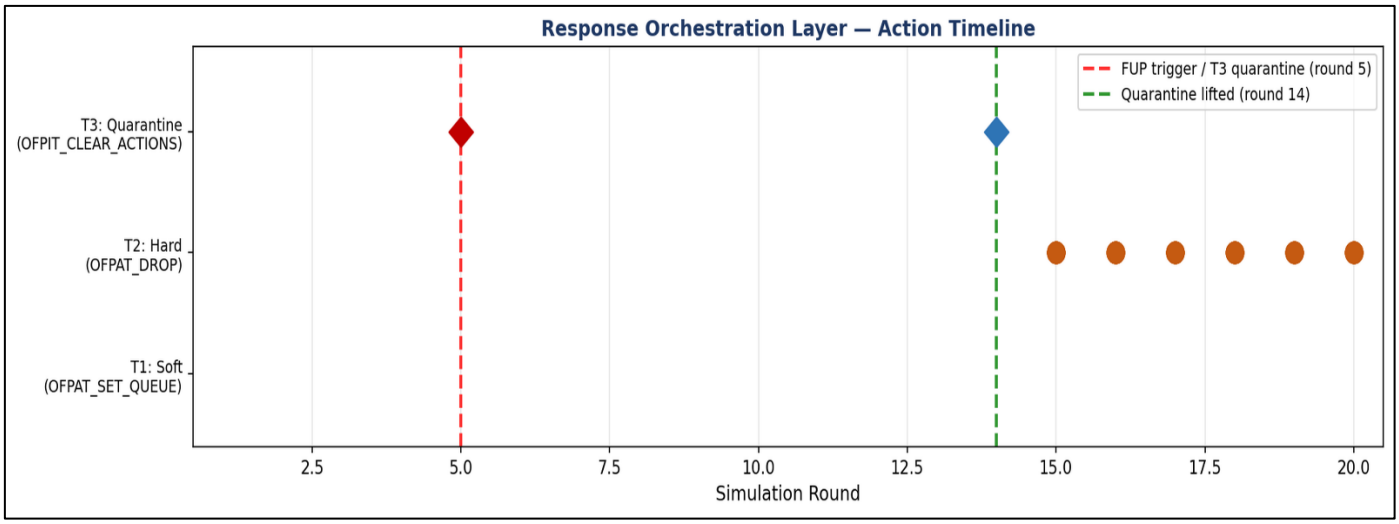

Figure 4. ROL action timeline over 20 rounds. T3 quarantine starts at round 5 and ends at round 14; T2 hard-drop rules activate from round 15, while T1 remains inactive

5. FEDERATED UNLEARNING EVALUATION

5.1 Dataset

We use the Mendeley SDN DDoS Dataset. Built from an emulated SDN environment and this is a publicly available benchmark. Before training, we apply IsolationForest outlier removal with a 10% contamination threshold and StandardScaler normalisation. After preprocessing, 93,910 records remain. Of these, 57,828 are benign (61.6%) and 36,082 are DDoS attack flows (38.4%). The dataset is moderately imbalanced, with benign traffic making up roughly two-thirds of all records.

Each record before pre-processing has 19 features. Three columns-source IP, destination IP, and timestamp-are dropped before training because they are not informative for classification. The rx kbps and tot kbps collapsed into byte rate. N flows and average duration features are renamed to flow count and average flow duration. Protocol is pre-computed to tcp ratio (fraction of flows using TCP), which carries the same discriminative signal for SYN-flood detection without a categorical encoding step.

Grouping features we have: Port-level features include port drop rate and Rx/Tx ratios. Flow-level features include flow count, average flow duration, short flow ratio and high packet flow ratio. Traffic volume features include packet rate and byte rate. Entropy features include source ip entropy and destination port entropy. Protocol features include tcp ratios. Controller features include packet in rate, table utilization and table miss rate. Finally, we have 14 features.

This feature schema matches exactly what FL-NDR's SCA collects from the OFPMP primitives. This confirms that the dataset is a realistic representation of SDN sensor output.

The 93,910 records are divided into 10 equal partitions of approximately 9,391 records each. One partition goes to each FL client. The split uses IID sampling. This means all clients see a similar mix of attack and benign traffic, so differences in unlearning performance are not caused by data distribution differences between clients. Each client uses 10% of its partition per round, which is approximately 939 records per round. The federated learning setup for the experiments is described in Table 2.

Table 2. Federated Unlearning Network Detection and Response (FL-NDR) Federated Learning (FL) and training configuration

Parameter	Value
FL clients / clients per round	10 / 3 (random, seed = 42)
Communication rounds	15 for S1 and S2 / 4 pre-trigger + up to 15 post-trigger for S3-S6
Model	MLP: 14 -> 288 -> 224 -> 1 (sigmoid), dropout = 0.105; binary cross-entropy loss
Optimiser / learning rate	Adam / 7.42×10^{-4} (weight decay 9.28×10^{-5})
Local epochs / batch size	10 / 107
Aggregation / payload	FedAvg / SafeTensors (0.28 MB per checkpoint)
Compromised client / trigger	Client 2 / end of round 5
Dataset	Mendeley SDN DDoS Dataset (93,910 records after preprocessing, IID partition)

Table 3. The six experimental scenarios (S1-S6)

ID	Label	Retraining	Quarantine	Description
S1	Baseline	-	-	Clean FL, no attack-15 rounds
S2	No Defence	-	-	Client 2 poisoned, no unlearning applied
S3	Full, No Quarantine	Full (round 0)	No	Full retraining from round-0 checkpoint
S4	Rapid, No Quarantine	Rapid (round 2)	No	Rapid retraining from round-2 checkpoint
S5	Full + Quarantine	Full (round 0)	Yes	Full retraining with Client 2 quarantined throughout
S6	Rapid + Quarantine	Rapid (round 2)	Yes	Rapid retraining with quarantine-best result

5.2 Threat model and configuration

There are two threats in this experiment. The first is a volumetric DDoS attack targeting Domain 2's SDN switches. The second is a model poisoning attack by Client 2. Client 2 uses a label-flipping attack: 50% of labels in its training data are flipped from round 5 onward. On the dataset's 61.6% benign / 38.4% attack distribution, a 50% random flip pushes the poisoned client's label distribution toward 50/50. This is detectable as an entropy increase compared to honest clients. The unlearning trigger fires at the end of round 5 in all scenarios S3 through S6.

5.3 Experimental scenarios

There are six scenarios in total. They are labelled S1 to S6. Table 3 shows the full matrix. S1 is the clean baseline. All 10 clients are honest and training runs for 15 rounds with no attack. This gives us the target accuracy to recover to.

Scenarios S2-S6 introduces model poisoning on Client 2 at round 5, which led to the automatic malicious client detection using entropy and z-score to automatically flag Client 2 as compromised at round 5. S2 does not perform any unlearning after the anomaly detection. This shows what happens when no defence is in place. S3-S6 triggers unlearning once the anomaly detector fires at round 5. S3 and S5 follows the Full Retraining strategy and starts retraining from round 0 onwards; whereas S4 and S6 follows the Rapid Retraining Strategy and starts retraining from the previous round of the compromised client's initial participation. (i.e. S4 and S6 starts again from round 2 because Client 2 initially participated in round 3). Further, S5 and S6 quarantines the compromised client until recovery.

Entropy's z-score values determine whether the client has recovered or not. Higher entropy value meant balanced distribution for both classes. However, selecting the client with the highest entropy value does not quantify if it is abnormal or slightly higher than its peers. Computing the signed z-score of the entropy values normalized it against the group distribution. This allowed selecting the outlier with significant statistical difference as the anomaly.

5.4 Results and analysis

S2-What happens with no defense. Accuracy drops to 93.59% and stays there for all 15 rounds. The nine honest clients keep pushing the model upward, but they cannot overcome one poisoned client under FedAvg. This shows that the majority cannot passively fix a poisoning attack on its own.

S3 and S4-Retraining without quarantine. Both scenarios end up around 93.7%. This is barely better than S2. The reason is straightforward. After the checkpoint rollback, Client 2 is still in the client pool. It gets re-selected in later rounds and reintroduces the poisoning signal. The rollback effect is gradually undone as shown in Figure 5. Retraining without quarantine gives no sustained improvement as shown in Table 4.

S5-Full retraining with quarantine. Accuracy reaches 93.71%. The quarantine stops Client 2 from re-poisoning during recovery. But the starting point is the problem. Resetting to round 0 throws away all benign-client learning. Starting from scratch with nine clients and 15 recovery rounds is not enough to close the gap.

S6-Rapid retraining with quarantine. Accuracy reaches 98.81%, matching the clean baseline within 0.21 pp. Starting from the round-2 checkpoint keeps what the honest clients already learned. With Client 2 excluded throughout, recovery follows the same trajectory as S1. The extra training time is 155 seconds with a 16.2% overhead compared to S2. S6 is also faster than S5 by 136 seconds while achieving 4.89 pp higher accuracy. It is the best configuration on both dimensions. Figure 6 presents the accuracy trajectory for each scenario.

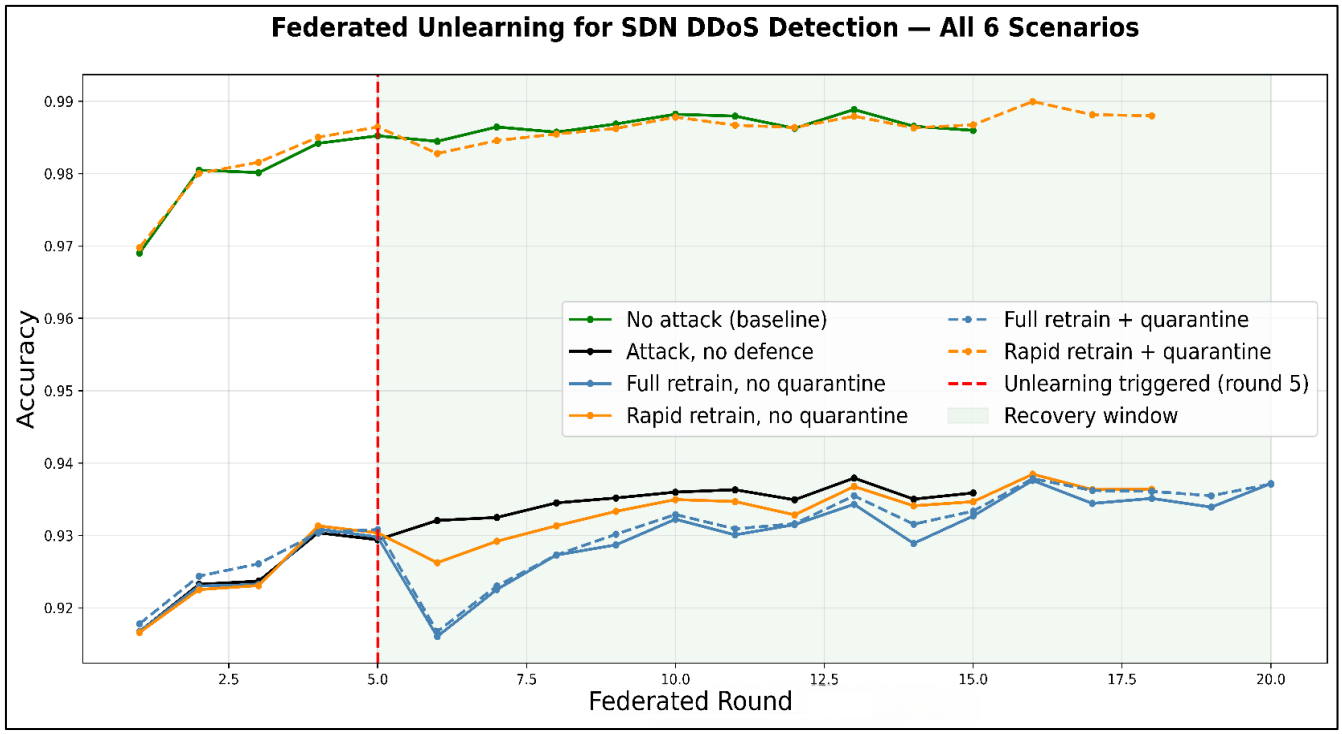

Figure 5. Accuracy trajectories for S1-S6: unlearning is triggered at round 5, only S6 returns to near-baseline accuracy

Table 4. Performance metrics and training time for all six scenarios

ID	Configuration	Accuracy	Precision	Recall	F1-Score	Confusion Matrix				FPR	FNR	ROC-AUC	Δ vs S1 (pp)	Elapsed Time (s)
						TN	FP	FN	TP					
S1	Baseline-clean	98.60%	0.9863	0.9860	0.9860	56634	1194	123	35959	0.0206	0.0034	0.9993	0.00	970.0
S2	Poisoning-no defence	93.59%	0.9358	0.9359	0.9359	53767	2948	3074	34121	0.0519	0.0826	0.9517	-5.01	951.6
S3	Full retrain, no quarantine	93.72%	0.9371	0.9372	0.9371	54016	2699	3201	33994	0.0475	0.0860	0.9519	-4.88	1278.0
S4	Rapid retrain, no quarantine	93.64%	0.9363	0.9364	0.9363	53851	2864	3110	34085	0.0504	0.0836	0.9519	-4.96	1123.1
S5	Full retrain + quarantine	93.71%	0.9370	0.9371	0.9370	54037	2678	3228	33967	0.0472	0.0867	0.9519	-4.89	1243.3
S6	Rapid retrain + quarantine	98.81%	0.9883	0.9881	0.9881	56851	977	143	35939	0.0168	0.0039	0.9992	+0.21	1107.0

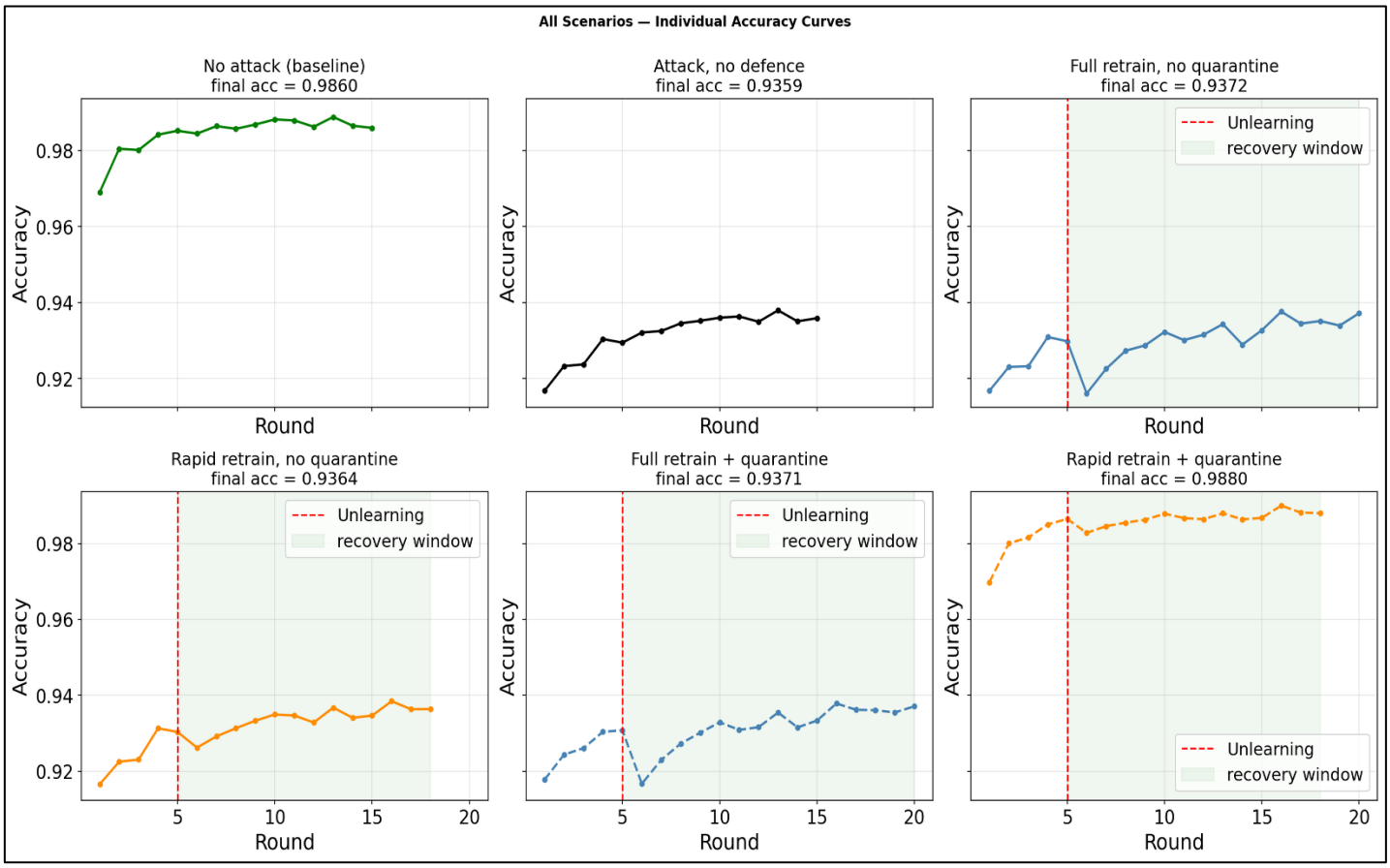

Figure 6. Individual accuracy trajectories by scenario

6. DISCUSSIONS

6.1 Quarantine is not optional

The most important finding is that quarantine is not a safety feature you can skip. It is in fact a necessary component of the unlearning process. Look at S4 and S6: both use rapid retraining from the same round-2. The only difference is quarantine. S4 ends at 93.64%. S6 ends at 98.81%. That is a 5.17 pp difference from one design choice.

Without the quarantine step, Client 2 continue to stay in the client pool. It gets re-selected during recovery rounds and sends poisoned updates again. Results from Table 4 shows that this process undo the benefit of the unlearning process. The same pattern holds for full retraining: S3 ends at 93.72% and S5 ends at 93.71%. The gap is only 0.01 pp because full retraining starts from random weights, so the checkpoint quality matters less and quarantine adds relatively little on top of it.

The main takeaway is that checkpoint selection and quarantine must be used together. A good checkpoint is only effective if the compromised client cannot re-enter the federation during recovery.

6.2 Why the sensor fabric matters

Every feature in the Mendeley SDN DDoS dataset comes from the same OpenFlow Statistics API that an SDN controller already uses to manage the network. There is no separate monitoring system to deploy. Any SDN network running OpenFlow can use FL-NDR's sensor layer with no hardware changes.

There is also a practical advantage during an attack. The polling rate is configurable. If the controller sees elevated

Packet-In rates from specific switches, it can increase the polling frequency for those switches. This gives finer-grained telemetry exactly when it is needed most.

6.3 Limitations

This evaluation uses a single compromised client. Gradient entropy signals and network-level divergence from the DSL are other natural candidates for automated client compromise detection, but we do not evaluate them here.

The data is distributed IID across clients. Mostly, real networks have different traffic profiles in each domain. Non-IID distributions may need FL variants such as FedProx to keep training stable during recovery. Multi-client poisoning and adaptive adversaries who spread their poisoning across rounds to avoid detection are also open problems.

The SDN simulation uses synthetic traffic. Future work should test the sensor layer against real SDN testbed traffic, for example from Mininet.

7. CONCLUSIONS

FL-NDR treats every OpenFlow switch in an SDN network as a sensor. Together, these switches form a distributed sensor layer that feeds data into a federated DDoS detection system. When a compromised FL client is discovered, FL-NDR rolls back the global model and installs OpenFlow quarantine rules at the same time. Both the model and the network are fixed at once.

The SDN simulation confirms that the OFPMP telemetry produces clear multi-feature DDoS signatures, and that the ROL correctly escalates from T2, the hard-drop to T3, the domain quarantine. The federated unlearning experiments done across the scenarios S1 to S6 show that rapid retraining from with quarantine (S6) is the only configuration that fully recovers baseline detection accuracy at 98.60%.

Unlearning without quarantine (S3, S4) gives no sustained improvement. Quarantine without a good checkpoint (S5) gives only a marginal improvement. Both must be used together. SDN is often described as a security risk because all intelligence lives in one controller. FL-NDR shows the other side of this: that same programmability makes it possible to respond to an attack faster and more precisely than any traditional network can.

REFERENCES

[1] Ahuja, N., Singal, G., Mukhopadhyay, D. (2020). DDOS attack SDN dataset. Mendeley Data, 1. https://doi.org/10.17632/jxpfjc64kr.1

[2] Yu, X.L., Wang, Z., Wang, M.M. (2026). A cost-efficient federated unlearning framework with rollback and compression optimization. Knowledge-Based Systems, 340: 115699. https://doi.org/10.1016/j.knosys.2026.115699

[3] Zaidoun, A.S., Lachiri, Z. (2025). A hybrid deep learning model for multi-class DDoS detection in SDN networks. Annals of Telecommunications, 80: 459-472. https://doi.org/10.1007/s12243-025-01085-1

[4] Najar, A.A., Naik, S.M., Lone, F.R., Nazir, A. (2025). A novel CNN-enhanced detection and mitigation of DDoS attacks in SDN. Cluster Computing, 28: 347. https://doi.org/10.1007/s10586-024-05003-3

[5] Bai, T.R., Liu, Y.W., Gao, Y.W., Zhou, Y.B. (2026). Ats-dta: Adaptive two-stage DDoS detection with dynamic threshold adjustment in SDN networks. Cybersecurity, 9: 12. https://doi.org/10.1186/s42400-025-00414-0

[6] Bahashwan, A.A., Anbar, M., Manickam, S., Bin-Salem, A.A. (2026). Deep learning-based detection mechanism for DDoS attacks targeting SDN controller. Journal of Network and Systems Management, 34: 76. https://doi.org/10.1007/s10922-026-10048-3

[7] Kalafy, S.A.A., Pashazadeh, S., Salehpour, P. (2025). Dynamic graph neural network-based framework to increase detection accuracy in SDN under DDOS. Scientific Reports, 16: 2305. https://doi.org/10.1038/s41598-025-32102-x

[8] Naeen, H.M., Ghadamyari, M., Barmar, M. (2025). Enhancing SDN security with deep learning and F-balanced cross-entropy for DDoS detection. Scientific Reports, 15: 33419. https://doi.org/10.1038/s41598-025-18826-w

[9] Elzoghbi, M., He, H. (2026). Kernel-level LDoS attack detection in SDN networks: An eBPF/XDP framework with dynamic thresholding. Computer Networks, 275: 111939. https://doi.org/10.1016/j.comnet.2025.111939

[10] Wang, J., Wang, L.P. (2025). LR-STGCN: Detecting and mitigating low-rate DDoS attacks in SDN based on spatial-temporal graph neural network. Computers & Security, 154: 104460. https://doi.org/10.1016/j.cose.2025.104460

[11] Alharbi, M., Haseeb, K., Humayun, M. (2025). AI-driven SDN and blockchain-based routing framework for scalable and trustworthy AIoT networks. Computer Modeling in Engineering & Sciences, 145(2): 2601-2616. https://doi.org/10.32604/cmes.2025.073039

[12] Macas, M., Wu, C., Fuertes, W. (2024). Adversarial examples: A survey of attacks and defenses in deep learning-enabled cybersecurity systems. Expert Systems With Applications, 283: 122223. https://doi.org/10.1016/j.eswa.2023.122223

[13] Kokila, M., Reddy, K.S. (2026). DeepSDN: Deep learning based software defined network model for cyberthreat detection in IoT network. ACM Transactions on Internet Technology, 26(1): 10. https://doi.org/10.1145/3737875

[14] Nazari, H., Yazdinejad, A., Dehghantanha, A., Zarrinkalam, F., Srivastava, G. (2024). P3GNN: A privacy-preserving provenance graph-based model for autonomous APT detection in software defined networking. Workshop on Autonomous Cybersecurity, 34-44. https://doi.org/10.1145/3689933.3690836

[15] Hormozi, M., Erfani, S.H., Sahafi, A., Moradi, M. (2026). Fed-adapt: A federated learning framework for adaptive topology reconfiguration against multi-rate DDoS and database flooding attacks. Journal of Information Security and Applications, 98: 104384. https://doi.org/10.1016/j.jisa.2026.104384

[16] Duy, P.T., Hien, D.T.T., Luong, T.D., Pham, V.H., Quyen, N.H. (2024). Fed-evolver: An automated evolving approach for federated intrusion detection system using adversarial autoencoder in SDN-enabled networks. Internet of Things, 28: 101397. https://doi.org/10.1016/j.iot.2024.101397

[17] Yasarathna, T.L., Le-Khac, N.A. (2025). SoK:

Systematic analysis of adversarial threats against deep learning approaches for autonomous anomaly detection systems in SDN-IoT networks. Journal of Information Security and Applications, 94: 104220. https://doi.org/10.1016/j.jisa.2025.104220

[18] Cai, L., Gu, K., Lei, J.Q. (2025). Defending federated learning system from poisoning attacks via efficient unlearning. Computers, Materials & Continua, 83(1): 239-258. https://doi.org/10.32604/cmc.2025.061377

[19] Gao, K., Zhu, T.Q., Ye, D.Y., Zhou, W.L. (2024). Defending against gradient inversion attacks in federated learning via statistical machine unlearning. Knowledge-Based Systems, 299: 111983. https://doi.org/10.1016/j.knosys.2024.111983

[20] Lu, Z.B., Zhang, X., Li, T., Wang, Y.L., Li, G.S., Liu, Z.Q. (2025). FeaUn: Feature unlearning in vertical federated learning for IIoT against feature inference attacks. Neurocomputing, 652: 131110. https://doi.org/10.1016/j.neucom.2025.131110

[21] Chao, J.Q., Li, X.K., Ai, Y.M., Qu, M.X. (2026). FedASU: Attention-guided sensitivity unlearning framework for multi-scenario federated graph unlearning. Expert Systems With Applications, 320: 132218. https://doi.org/10.1016/j.eswa.2026.132218

[22] Usmani, M.M.A., Tahir, M.A., Faisal, H., Rafi, M. (2026). Federated unlearning using diffusive noise injection. Information Fusion, 125: 103796. https://doi.org/10.1016/j.inffus.2025.103796

[23] Wang, C., Chen, J., Yang, Y., Ma, X.Q., Liu, J.C. (2022). Poisoning attacks and countermeasures in intelligent networks: Status quo and prospects. Digital Communications and Networks, 8(2): 225-234. https://doi.org/10.1016/j.dcan.2021.07.009

[24] Li, B.C., Su, N.X., Ying, C., Wang, F. (2023). Plato: An open-source research framework for production federated learning. In Proceedings of the ACM Turing Award Celebration Conference, China, pp. 1-2. https://doi.org/10.1145/3603165.3607364

[25] TL-System. (2026). Plato: A research framework for federated learning. GitHub repository. https://github.com/TL-System/plato.