engineering proceedings

MDPI

Article

Online Q-Learning for Autonomous LDoS Detection and Mitigation in P4-Based SDN †

Alizon Valdiviezo * and Christian Tipantuña *

Department of Electronics, Telecommunications and Computer Networks, Escuela Politécnica Nacional, Quito 170525, Ecuador

* Correspondence: alizon.valdiviezo@epn.edu.ec (A.V.); christian.tipantuna@epn.edu.ec (C.T.)

† Presented at the XXXIV Conference on Electrical and Electronic Engineering (JIEE 2026), Quito, Ecuador, 26–30 October 2026.

Abstract

Low-rate denial-of-service (LDoS) attacks often go undetected because their periodic, intermittent behavior initially resembles legitimate traffic. Existing machine learning methods also offer limited help; they depend on labeled datasets and offline training, which are not available before an attack occurs. This paper proposes a Q-learning agent that learns online on the network without a pre-collected dataset, over a data plane programmed in P4 (Programming Protocol-Independent Packet Processors). Traffic monitoring uses a per-flow Count-Min Sketch; in-band network telemetry (INT) activates only when anomalies appear, keeping overhead low during normal operation. When the agent decides to act, it identifies the attacking host and installs a drop rule on the P4 switch through the Thrift interface, without human intervention. The system is evaluated over 21 consecutive training sessions with the Q-table persistent between sessions; over the nine sessions following the onset of the low-exploration phase (n = 2004 decisions), the average F1-score was 0.855 (95% CI: 0.809–0.901), the false positive rate 0.141 (95% CI: 0.090–0.192), and the detection latency 4.21 s (95% CI: 3.08–5.34 s). The F1-score and the detection latency meet their target thresholds (F1-score > 0.70, detection latency < 10 s) across their entire confidence intervals. The false positive rate meets its target (<0.15) on average, but the upper bound of its interval (0.192) exceeds it, so this target is not met robustly. These results suggest that online learning is a viable approach for autonomous defense in programmable networks.

Keywords: RL; LDoS; P4; Q-learning; SDN; telemetry

1. Introduction

Low-rate denial-of-service (LDoS) attacks do not produce large volumes of traffic. Instead, they send short, periodic bursts that, on average, resemble legitimate traffic [1]. Conventional detection mechanisms that monitor total volume fail to detect them. However, the damage is real, as they degrade service and consume target resources over time. This limitation is more consequential in software-defined networks (SDNs), where control is centralized in a single controller; therefore, an attacker needs only to saturate the communication channel between that controller and the switches to paralyze the network [2]. Increasing the polling frequency does not resolve this limitation either, because if the controller polls the switches frequently enough to detect a low-rate pattern, it saturates that channel before the attack is identified [3]. Because neither volume-based detection nor frequent queries are feasible, identifying an LDoS attack requires analyzing traffic evolution

Check for updates

Academic Editor: Diego Echeverría

Published: 2 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

rather than a single point in time. Yungaicela-Naula [4] applied this principle and demonstrated that deep reinforcement learning (DRL) models can mitigate low-rate attacks in OpenFlow-based SDN. However, these models rely on labeled data collected before deployment (data that dynamic production environments rarely provide). This limitation leads to the question addressed in this work, namely whether an agent can learn to distinguish attack traffic from normal traffic while operating on the network, without prior access to labeled examples [5].

To address this problem, this paper presents a prototype built on the system of [6]. It combines a Count-Min Sketch (CMS) with conditional activation of in-band network telemetry (INT) in the data plane using P4, which reduces overhead during normal operation. It also includes an online Q-learning agent that, upon detecting an attack, installs drop rules directly into the P4 switch via the Thrift interface without human intervention. The main contribution of this paper is an experimental analysis spanning 21 training sessions, in which the low-exploration phase is identified and compared with predefined target thresholds.

Relation to Prior Work

This manuscript builds on the system prototype developed in the first author's undergraduate thesis, supervised by the second author [6]. The architecture is the same, namely the Count-Min Sketch with conditional INT, the three-action tabular agent, the reward function, the blocking hysteresis, and the ground-truth labeling. Two controller parameters changed, as summarized in Table 1: the discretization of the variance feature and the epsilon decay schedule. The evaluation also changed. The results reported here come from new runs that start with an empty Q-table and are computed only over the low-exploration phase, whereas [6] reports all valid sessions, including those with exploration. The results in Section 4 are therefore new and should not be compared directly with those of [6]. An attempt to refine the discretization of Δ_{max} did not improve the F1-score and is discussed in Section 4.

Table 1. Differences between the degree project [6] and this manuscript. All other components are unchanged.

Element	Prior Work [6]	This Manuscript
Variance discretization	5 levels of width 50	12 levels, finer between 11 and 45 (state space from 2100 to 5040)
Epsilon decay factor	0.99; $\epsilon = 0.05$ reached within the first session	0.999186; $\epsilon = 0.05$ reached during session 10
Q-table	From the earlier runs	New, started empty
Evaluated sessions	All valid sessions (20, $n = 4369$)	Low-exploration phase (9 sessions, $n = 2004$)

2. Related Work

Considerable research addresses SDN security against denial-of-service attacks, mostly focusing on high-rate volumetric threats, while low-rate attacks remain underexplored. Shifting detection to the P4 data plane reduces control plane reliance at line rate. A systematic review [7] categorizes existing detection methods into statistical, machine learning (ML), and deep learning (DL) approaches, identifying limited coverage of low-rate attacks as a key gap. Low-rate attacks in programmable networks are surveyed in [8], and machine learning detectors against them are reviewed in [9].

Data-plane solutions have addressed volumetric traffic. For example, ref. [10] uses HyperLogLog and Count-Min Sketch in P4 to trigger controller alerts; ref. [11] combines Count-Min Sketch asymmetry detection with a controller CNN classifier; and ref. [12] uses attack graphs to restrict entropy calculations, achieving low detection times. However, these approaches rely on static thresholds, entropy baselines, or offline-trained classifiers that do not adapt dynamically.

Feature-based mechanisms enhance accuracy but still depend on offline processing. In [13], traffic features extracted from P4 are classified via SVM and Random Forest. Random Forest, SVM, and K-nearest neighbors classifiers have also been trained on features extracted in a stateful P4 data plane, for TCP flood attacks [14]. Similarly, ref. [15] uses supervised models for monitoring thresholds, highlighting dependence on labeled data and pointing to reinforcement learning (RL) as a future direction. RL in P4 networks has primarily tuned parameters rather than detecting attacks without labels. DESiRED [5] employs a DQN agent for queue management via in-band telemetry, but lacks cross-topology generalization. Online classifiers have been applied to low-rate attacks in SDN, for example a Passive-Aggressive model that updates incrementally in the controller [16]. Reinforcement learning has also been used against flooding. A DDPG agent learns bandwidth limits [17], and a DDQN agent limits and redirects flows while preserving quality of service, relying on an external detector to separate benign from malicious flows [18]. A lightweight online reinforcement learning approach for autonomous defense against network attacks in SDN is reported in [19]. Most relevantly, ref. [4] combines LSTM and DRL to mitigate low-intensity DDoS attacks in ONOS. Nevertheless, detection remains in the controller and still requires pre-collected data prior to deployment, leaving autonomous data-plane online detection unresolved.

Comparison with Other Approaches

Table 2 places the proposed agent among the main families of methods discussed above. The comparison is qualitative and rests on what the cited works describe. Section 4 adds an offline replay of two statistical detectors on the logs of this work.

Table 2. Qualitative comparison with other detection and mitigation approaches.

Approach	Labeled Data	Adaptation	Cost and Limitation
Supervised ML/DL [9,14,20]	Labeled dataset for offline training	Retraining when traffic changes	Training cost and manual feature engineering [9]
Online ML [16]	Labeled instances to update the model	Incremental updates	Evaluated on labeled datasets in emulation
Deep RL [4,17,18]	Labels or an external detector to compute the reward during training	Policy learned before deployment	Neural network policy and training episodes; mostly volumetric DDoS
Rules and statistics [7,10,12]	No	Only by retuning thresholds or baselines	Static thresholds or entropy baselines
This work	No pre-collected dataset, and the reward is computed from a fixed threshold on `max_est` (Section 3.3) without external labels	Online, Q-table updated every window	5040 discretized states, INT overhead of 5.22%; compared with two statistical detectors in an offline replay (Section 4)

The main difference with supervised and online ML is that the agent needs no dataset collected and labeled beforehand and has no offline training phase. This does not remove the need for a feedback signal, since the reward relies on a fixed threshold on the sketch signal `max_est` (Section 3.3), which is also used by the ground-truth engine described in Section 3.6. Deep RL methods [17,18] also learn from interaction, but they rely on neural network policies and a training stage before deployment, whereas the tabular agent keeps a Q-table over 5040 discretized states. Some statistical detectors also adapt their reference to recent traffic [21,22], but they rely on parameters set beforehand. No supervised classifier or other RL agent was run in the same P4 and LDoS scenario, and no published comparison that does so was found. Section 4 compares the agent with two statistical detectors replayed on the same logs, so the trade-off between accuracy and cost against supervised and RL methods is not quantified here.

3. Proposed Solution

Section 2 left an important question unresolved, namely whether an agent can autonomously distinguish attack traffic from normal traffic in real time, without access to previously collected labeled data. Addressing this question necessitates a prototype capable of continuously monitoring network traffic, determining whether to permit, scrutinize, or block it, and executing these actions autonomously. This section presents such a prototype, comprising a closed-loop system between the P4 data plane, which estimates traffic behavior at line speed, and a custom SDN controller that hosts an online Q-learning agent and communicates its decisions to the switch through a Thrift-based channel.

3.1. Overview of the System Prototype

The prototype implements the standard three-plane SDN architecture. The management plane serves as the experimental environment responsible for orchestrating traffic generation and session logging. This section examines the data and control planes, which together establish the closed decision loop. The data plane comprises six BMv2 switches configured in a three-tier topology with a central switch (S1), two distribution switches, and three access switches (Figure 1). Both the victim and the attacker are connected directly to S1. The control plane consists of a Python-based controller that hosts the Q-learning agent and communicates with S1 via Thrift, the interface provided by BMv2 for external processes to read and modify switch state. Traffic is processed continuously at the data plane, with every packet updating the CMS registers, independent of controller activity. The controller periodically polls the data plane; every `INTERVAL` = 5 s, it reads the registers via Thrift, resets them, and converts the raw counts into four features that characterize the network state during that interval. The agent then discretizes the state and selects one of three possible actions, and the controller implements the chosen action, as depicted in Figure 2. Each session consists of an initial 200 s baseline of legitimate traffic (ICMP, UDP, TCP, HTTP, and SNMP flows), followed by five 150 s Shrew-style attack bursts separated by 80 s recovery intervals. The tools and rates used to generate each flow are listed in Table 3, and the network and link parameters in Table 4.

The prototype was implemented and evaluated using BMv2 1.15.0 (v1model), the $P4_{16}$ language compiled with p4c 1.2.5.10, Mininet 2.3.1b4, Scapy 2.5.0, and Python 3.12. No random seed was fixed. Randomness enters through the ϵ-greedy exploration of the agent and through the source and destination ports of the legitimate traffic generator, while the attack generator is deterministic. A new run of the experiment therefore gives comparable but not identical results, whereas the analysis can be reproduced exactly from the released decision log (see the Data Availability Statement).

https://doi.org/10.3390/engproc2026158033

Table 3. Traffic generation parameters (`attack_shrew.py` and `legitimate_traffic.py`). All packets are built with Scapy and sent at layer 2. Legitimate flows go from each source host to h1. Rates are the nominal script parameters.

Flow	Source	Packet	Rate
Shrew attack	h2	TCP SYN to 10.0.1.1, port 80	Bursts of 0.2 s at 300 pps (60 packets), one burst every 1.0 s
ICMP	h3	Echo request	1 pps
UDP	h4	Random source and destination ports (1024 to 65,535)	1 pps
TCP	h5	SYN to a random port in {80, 443, 8080, 22, 21}	1 pps
HTTP	h6	GET/over TCP, port 80	1 pps
SNMP	h7	UDP to port 161	1 pps

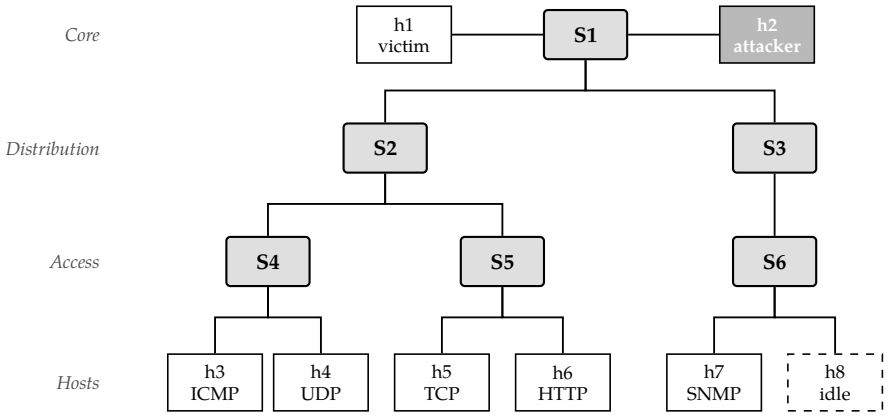


Figure 1. Emulated topology with six P4 (BMv2) switches in three tiers and eight hosts. h1 is the victim, h2 the LDoS attacker, and h3–h7 each generate one legitimate traffic type toward h1. h8 is attached to S6 but generates no traffic.

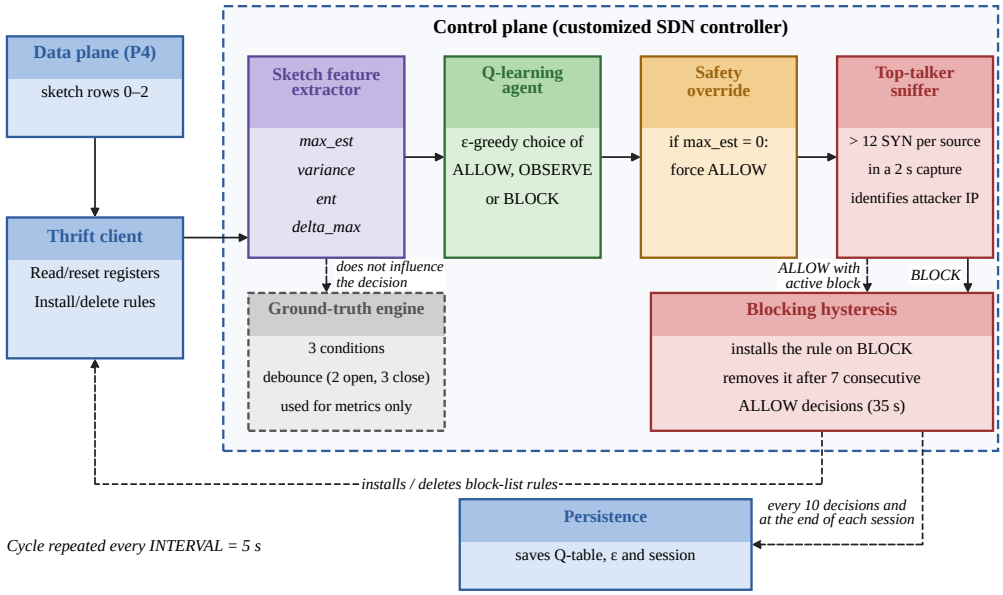


Figure 2. Closed-loop decision cycle of the prototype. Solid arrows indicate the main flow of data and decisions, and dashed arrows indicate secondary paths, such as evaluation, rule installation and removal, and saving of the agent state. Colors distinguish component types: blue for the interface with the switch and the saved state, purple for feature extraction, green for the agent, amber for the safety control, red for mitigation, and gray for the evaluation-only engine.

Table 4. Network and link parameters of the emulated topology. Round-trip times are the mean of 50 ICMP echo requests (minimum–maximum in parentheses).

Parameter	Value
Switches/hosts	6 (BMv2 1.15.0, v1model)/8
Link bandwidth	No rate limit configured
Configured link delay	0 ms
BMv2 egress queue	64 packets per port (default)
Mininet (netem) queue	1000 packets per interface (default)
RTT h1–h2 (1 switch)	4.42 ms (3.15–6.92)
RTT h1–h3 (3 switches)	13.2 ms (9.57–19.3)

3.2. Adaptive Telemetry in the P4 Data Plane

This approach addresses the overhead problem described in Section 1. A telemetry scheme that inspects every packet in detail can saturate the same control channel targeted by an LDoS attack. To mitigate this, the data plane employs two mechanisms, a lightweight one and a more resource-intensive one. The system defaults to the lightweight mechanism and only activates the heavier process when the first mechanism identifies anomalous activity. The lightweight mechanism is a CMS, which uses three parallel registers, each containing 1024 cells (`sketch_row0` to `row2`). Each packet is hashed using three hash functions (CRC16, CRC32, and a 16-bit checksum) applied to the source IP, destination IP, and transport protocol. The corresponding cell in each row is incremented without controller intervention, and the registers are read and reset at every decision window. The estimated traffic for a flow is determined by the minimum value among the three counters, which reduces the impact of hash collisions. This estimate never falls below the true count. For a sketch with w cells per row and d rows, it exceeds the true count by at most εN with probability at least $1 - \delta$:

$$\hat{f} - f \leq \varepsilon N \quad \text{with probability} \geq 1 - \delta, \qquad \varepsilon = \frac{e}{w}, \quad \delta = e^{-d}, \tag{1}$$

where f is the true count of a flow, $\hat{f}$ is its estimate, and N is the number of packets counted in the window [23].

With $w = 1024$ and $d = 3$, $\varepsilon \approx 0.0027$ and $\delta \approx 0.05$, the largest window in the evaluated sessions held $N = 268$ packets, which gives a maximum error of 0.71 packets, below the narrowest bin of `max_est` (3 packets, Table 5). The bound assumes pairwise independent hash functions, and the CRC16, CRC32, and checksum functions available in v1model do not provide that guarantee, so it is a reference value and not a strict guarantee. Because this process operates at line speed, its resource cost depends on the packets already being forwarded by the switch, rather than the number of flows monitored by the controller. The more resource-intensive mechanism is in-band network telemetry (INT), which inserts per-hop metadata into packet headers. INT provides significantly more information than CMS but incurs a higher per-packet cost, so it is disabled by default using a register called `int_mode_flag`. The controller enables INT only when the agent selects an action other than ALLOW. There is no separate threshold or statistical test for this decision. It follows the policy learned over the discretized state (Section 3.3), and exploratory actions can also enable INT. The controller also forces ALLOW when a window contains no packets. As long as network conditions remain normal, only the CMS is updated. This conditional activation enables the scheme to be adaptive, as the switch incurs the cost of detailed visibility only when necessary.

3.3. Online Q-Learning Agent

For each `INTERVAL`, the controller reads and resets the three CMS rows via Thrift, ensuring that each reading reflects only the current 5-s window rather than a cumulative total. From the raw counts, four features are computed, namely the maximum estimated packet count per 5-s window across flows (`max_est`), the variance across non-zero cells (`variance`), the entropy of the traffic distribution (`ent`), and the change in `max_est` from the previous interval (`delta_max`). As these features are continuous, they are discretized into a limited number of levels before being used as state variables, a standard procedure in tabular reinforcement learning to maintain a finite Q-table [24]. The discretization of `max_est`, `variance`, and `ent` employs variable interval widths, with narrower intervals where normal and attack traffic overlap and wider intervals elsewhere, to preserve the distinction in regions where the two classes are most difficult to differentiate. Table 5 gives the number of levels and the bin boundaries of each feature.

The cut points were set from the distribution of each feature in attack steps and in normal steps (labelled with the instantaneous ground truth) in earlier pilot sessions, using the minimum, quartiles, 90th percentile, and maximum of each class. The boundaries were then chosen by hand inside the region where the two classes overlap, not computed by an algorithm. For `max_est`, normal steps reached at most 25 (90th percentile 14) while attack steps started at 18 (8 sessions, 276 attack and 1410 normal steps), so the interval from 10 to 30 uses bins of width 3. For `variance`, attack steps never fell below 11.0 and normal steps never exceeded 43.2 (17 sessions, 1031 attack and 2556 normal steps), so values below 11 share one bin, the interval from 11 to 45 uses width 5, and larger values use width 75. For `ent`, attack steps started at 4.37 and normal steps reached 5.00 (7 sessions, 277 attack and 1232 normal steps), and the previous discretization placed 98.8% of the steps in a single bin, so the range from 4.3 to 5.0 is split into bins of width 0.14. The pilot sessions are not the sessions evaluated in Section 4. `delta_max` keeps a uniform width of 10 on its absolute value, because a finer split lowered the F1-score to 0.611.

Table 5. State discretization. The state is the tuple of the four levels (5040 possible states).

Feature	Levels	Bin Boundaries
`max_est`	12	0: < 10. 1 to 7: width 3 on $[10, 30)$. 8 to 11: $[30, 45)$, $[45, 60)$, $[60, 75)$, ≥ 75
`variance`	12	0: < 11. 1 to 7: width 5 on $[11, 45)$. 8 to 11: $[45, 120)$, $[120, 195)$, $[195, 270)$, ≥ 270
`ent`	7	0: empty window. 1: $(0, 4.3)$. 2 to 6: width 0.14 from 4.3, the last bin being ≥ 4.86
`delta_max`	5	$\lvert\Delta_{\max}\rvert$ with width 10: $[0, 10)$, $[10, 20)$, $[20, 30)$, $[30, 40)$, ≥ 40

The resulting state, represented as a tuple of the four discretized levels, directly indexes the Q-table. For each state, the agent selects one of three actions using an epsilon-greedy policy. ALLOW leaves traffic unchanged, OBSERVE (implemented as `RATE_LIMIT` in the controller's code) only enables INT for external visibility without blocking, and BLOCK installs a drop rule for the identified attacker. OBSERVE therefore gives the agent a cautious intermediate option between inaction and outright blocking. As a safety measure, if a window contains no packets (`max_est` $= 0$), the controller overrides the agent and enforces ALLOW, thereby preventing periods of inactivity from being misclassified as attacks. The agent is trained using a reward function that incorporates a deliberate asymmetry, as detailed in Table 6. The condition labeled "No relevant traffic" in that table corresponds to the same `max_est` $= 0$ case described above, for which ALLOW is always enforced regardless of the agent's own choice. Specifically, the penalty for mistakenly blocking legitimate traffic is

nearly twice as severe as the penalty for allowing an attack to proceed for one interval. This approach is based on the observation that an unblocked attack packet results in a maximum delay of 5 s, whereas a legitimate user who is wrongly blocked experiences a complete loss of service, which ultimately determines system usability. Assigning a higher cost to the more detrimental error is a standard practice in reward shaping [25], guiding the learned policy to prioritize service availability over aggressive blocking. Following each action, the agent updates $Q(s, a)$ using the Bellman equation, balancing the immediate reward with the optimal expected value of the subsequent state. The learning rate and discount factor are set to standard tabular Q-learning values ($lr = 0.1$, $\gamma = 0.95$) [24]. In contrast, the epsilon schedule was intentionally tuned. After each Q-value update, that is, once per decision, ϵ decays geometrically down to a floor:

$$\epsilon_{t+1} = \max(\epsilon_{\min},\, \lambda\, \epsilon_t), \qquad \epsilon_0 = 0.30, \quad \epsilon_{\min} = 0.05, \quad \lambda = 0.999186, \tag{2}$$

which reaches the floor after about 2200 decisions, during session 10, rather than within the initial session. This approach allows the agent greater opportunity for exploration before entering the low-exploration phase, which in Section 4 starts at session 13, after two full sessions at the floor. The values of Q, ϵ, and the session counter are saved to disk between sessions, ensuring that training is cumulative rather than restarting with each run.

Table 6. Reward function of the Q-learning agent.

Condition	Action	Reward
No relevant traffic	ALLOW	+2
No relevant traffic	OBSERVE/BLOCK	−5
Attack (`max_est` > 25)	BLOCK (new)	+10
Attack (`max_est` > 25)	BLOCK (active)	+3
Attack (`max_est` > 25)	OBSERVE	+6
Attack (`max_est` > 25)	ALLOW	−8
Normal, block active	ALLOW	+4
Normal, no block	ALLOW	+2
Normal traffic	BLOCK	−15
Normal traffic	OBSERVE	−2

3.4. Attacker Identification and Hysteresis-Based Mitigation

The CMS notifies the agent of anomalous traffic but does not identify the responsible host, as it aggregates all flows into shared counters. When the agent selects BLOCK, the controller initiates a two-second SYN-packet capture on S1's interfaces using Scapy, counting SYN-flagged packets per source IP address. Any address exceeding 12 SYN packets within this window is classified as the attacker, a threshold established based on the approximately 6 SYN packets generated by legitimate hosts in the same interval. This value is sufficient to distinguish the Shrew attack's bursts from normal background traffic. The controller subsequently installs a drop rule for the identified address via Thrift. Given the intermittent nature of the Shrew attack, immediately removing the rule when the agent selects ALLOW would result in repeated installation and removal during each silent gap between bursts. To address this, the rule is only removed after `ALLOW_HYSTERESIS` = 7 consecutive ALLOW intervals, corresponding to 35 s of sustained normal traffic. This approach enables the system to treat a burst attack as a single episode rather than a series of unnecessary block and unblock cycles.

3.5. Closed-Loop Decision Algorithm

Algorithm 1 integrates these steps into a single cycle executed by the controller at each `INTERVAL`: reading telemetry, discretizing the state, selecting and executing an action,

identifying the attacker when necessary, applying hysteresis, and updating the policy. Four thresholds are established for evaluation, primarily based on the system's parameters. Detection latency must remain below 10 s, which is twice the `INTERVAL`. Mitigation latency must remain below 75 s, corresponding to half the duration of an attack burst (150 s). The F1-score and false positive rate targets (0.70 and 0.15, respectively) are set with the recognition that perfect performance is unattainable in dataset-free online detection, where the agent lacks prior labeled examples for learning.

Algorithm 1: Online adaptive telemetry-driven mitigation

Data: Q, ϵ, `INTERVAL`, `ALLOW_HYSTERESIS`
Result: Updated Q-table and P4 mitigation state
allowStreak $\leftarrow 0$; *blockedIP* $\leftarrow \emptyset$
while *session is active* **do**
  $s \leftarrow$ ExtractState(`INTERVAL`); $a \leftarrow$ SelectAction(Q, s, ϵ)
  if $s.maxEst = 0$ **then**
    $a \leftarrow$ ALLOW // Safety override
  end
  if $a =$ *BLOCK* ***and*** *blockedIP* $= \emptyset$ **then**
    blockedIP $\leftarrow$ BlockTopTalker(); *allowStreak* $\leftarrow 0$
  else if $a =$ *OBSERVE* **then**
    EnableINT()
  else
    DisableINT()
    if *blockedIP* $\neq \emptyset$ **then**
      allowStreak $\leftarrow$ *allowStreak* $+ 1$; **if** *allowStreak* $\geq$ *ALLOW_HYSTERESIS* **then**
        ReleaseBlock(*blockedIP*)
      end
    end
  $r \leftarrow$ ComputeReward$(s, a,$ *blockedIP*$)$; UpdateQ(Q, s, a, r)
  $\epsilon \leftarrow$ Decay(ϵ) // Equation (2)
  PersistState()
end

3.6. Ground-Truth Engine

The `GroundTruthEngine` (Figure 2) labels each 5 s window as attack or normal, so that the F1-score, the FPR, and the attack episodes can be computed. It is used only for evaluation, the agent never receives its output, and neither the state nor the reward depends on it. The label is computed online from the same four features that the controller reads from the sketch, without access to the attack schedule. A window is labelled as attack if at least one of three conditions holds: (A) $|\Delta_{\max}| > 15$ and `variance` > 10, which captures the jump between windows at the start of a burst; (B) `max_est` > 25 and `variance` > 10, which captures a window containing a full burst; or (C) `ent` < 3.5 and $\Delta_{\max} > 10$, which captures a single flow that dominates and is growing. This instantaneous label is the one used to count true and false positives and negatives, and attack episodes, where an episode is a run of consecutive attack windows (Section 4). For mitigation latency, the engine applies a debounce. An episode opens after two consecutive attack windows and closes after three consecutive normal windows, so that the pulsating bursts of a Shrew attack count as one episode instead of several.

The label is derived from the same sketch signals that feed the agent, and the `max_est` threshold of condition B is the one used in the reward function (Table 6). It is therefore an automatic reference signal and not an independent measurement, which limits how the reported F1-score and FPR should be read. Section 4 therefore also reports the agent and

two statistical detectors scored against the known attack schedule, which does not depend on the sketch features.

4. Test and Results

The prototype was evaluated over 21 consecutive training sessions using a persistent Q-table. One session (original session 15, from a series of 22 runs) was excluded due to a failure in generating attack traffic, which was unrelated to the agent's behavior. Sessions are numbered consecutively in this paper, and the mapping to the original run numbers is provided in the repository [26]. The value of ϵ reached its minimum (0.05) during session 10, and the low-exploration phase analyzed below starts at session 13. Figure 3 presents the F1-score, false positive rate (FPR), Q-state growth, ϵ decay, cumulative reward, and detection and mitigation latency throughout the training sessions. Table 7 provides a summary of the results for the nine sessions in the low-exploration phase ($n = 2004$ decisions). Attack episodes are counted on the raw ground-truth signal (Section 3.6). Each session schedules five bursts, which gives 45 bursts in the nine sessions; every burst produces at least one episode, and nine bursts are split in two when the signal drops for a few windows and returns, which gives the 54 episodes in Table 7.

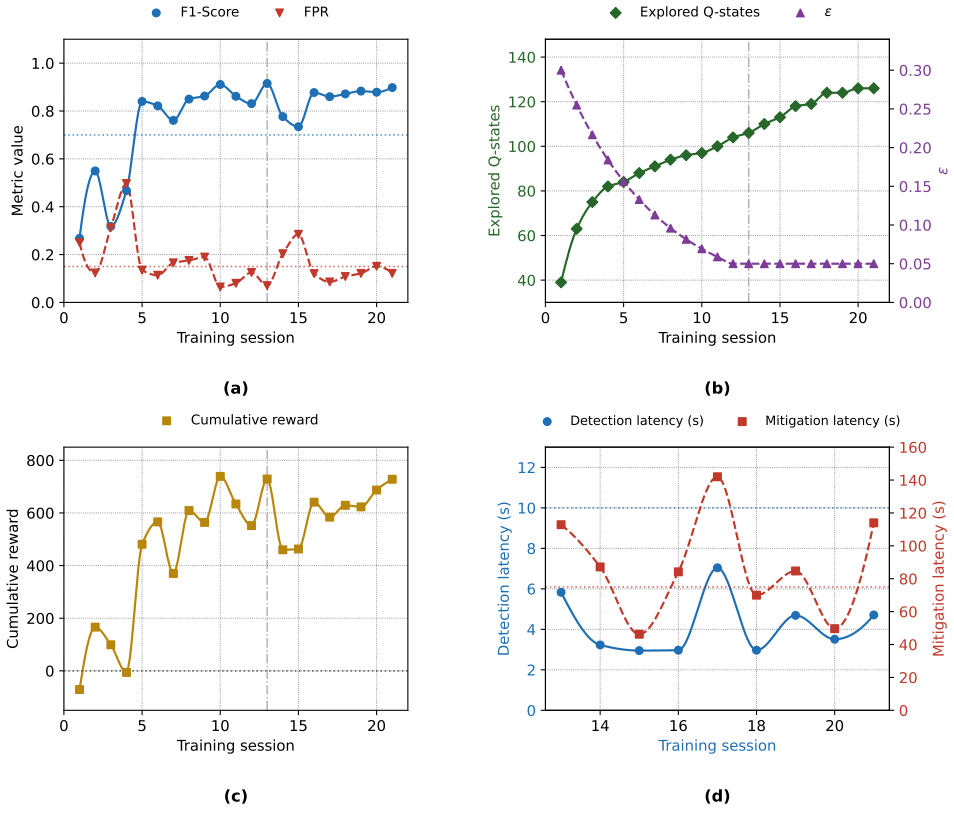


Figure 3. System metrics across the training sessions. (**a**) F1-score and FPR, plotted against their 0.70/0.15 thresholds, across all 21 sessions. (**b**) Explored Q-states alongside ϵ decay, across all 21 sessions. (**c**) Cumulative reward per session, across all 21 sessions. (**d**) Detection and mitigation latency in the low-exploration phase, against their 10 s/75 s thresholds. The dash-dotted line in (**a**–**c**) marks session 13, during which the low-exploration phase occurs. Dotted horizontal lines indicate the thresholds in (**a**,**d**), colored as their metric, and the zero reference in (**c**).

The mean F1-score (0.855) surpasses the 0.70 threshold, and the lower confidence bound also exceeds this benchmark. The FPR meets the 0.15 threshold on average, despite the variability characteristic of online learning. However, the upper limit of its 95% confidence interval (0.192) exceeds 0.15, so this target is not met robustly across sessions.

At the episode level, the agent detected 94.4% of the 54 recorded attack episodes. Detection latency (4.21 s) remained well below the 10 s threshold, while mitigation latency (82.69 s) exceeded the 75 s reference due to the seven-interval hysteresis, which prioritizes stability over speed (Section 3.4). Operational cost was stable (5.22% overhead), with minimal impact on legitimate traffic. Eight of nine sessions exhibited zero or near-zero packet loss (at most 0.03%), with one exception (S15, 5.18%). Refining the discretization of Δ_{max} using the same percentile method as for `max_est` and `variance` (Section 3.3) reduced the F1-score to 0.611, indicating that the tabular approach has a practical granularity limit.

Table 7. Results from the low-exploration-phase sessions ($n = 9$).

Metric	Value
F1-score (aggregate)	0.858
F1-score (mean, 95% CI)	0.855 [0.809–0.901]
False positive rate (aggregate)	0.144
False positive rate (mean, 95% CI)	0.141 [0.090–0.192]
Attack episodes detected	51/54 (94.4%)
Detection latency (s)	4.21 [3.08–5.34]
Mitigation latency (s)	82.69 [57.83–107.55]
Control-plane overhead (%)	5.22 [4.99–5.44]
Packet loss (legitimate traffic)	0.00–5.18%
Legitimate traffic throughput (kbps)	5.2
Cumulative reward per session	616.0 [539.1–692.9]
Q-states explored (range, S13–S21)	106–126

4.1. Comparison with Two Statistical Detectors

Two statistical detectors were replayed offline on the decision windows logged in the nine low-exploration sessions, so all methods use the same data. The first is an adaptive variance threshold, related to the reference-interval approach used for cyclical low-rate DDoS attacks in [21]. It raises an alarm when `variance` exceeds its exponentially weighted moving average by more than k standard deviations. The second is an entropy deviation detector, in line with the EWMA-updated entropy threshold of [22]. It raises an alarm when `ent` falls below its moving average by more than k standard deviations. Both ignore windows without packets, as the agent does. The value of k and the smoothing factor were chosen on sessions 1 to 12 to maximize the F1-score, and the detectors were evaluated on sessions 13 to 21.

The ground-truth engine builds its label partly from `variance` and `ent`, which are the features these detectors read, so scoring them against it would favor them. All methods were therefore scored against the known attack schedule, where a window is positive if it overlaps a burst. With this reference the agent reaches an F1-score of 0.825 and an FPR of 0.131 (Table 8). These values differ from Table 7 only because the reference differs, and they lead to the same conclusions on the 0.70 and 0.15 thresholds.

Table 8. Offline replay of two statistical detectors on the low-exploration sessions ($n = 9$), scored against the known attack schedule. Mean per session and 95% confidence interval.

Method	F1-Score	FPR
Q-learning agent	0.825 [0.779–0.871]	0.131 [0.083–0.180]
Adaptive variance threshold	0.915 [0.883–0.948]	0.043 [0.031–0.054]
Entropy deviation	0.873 [0.843–0.904]	0.286 [0.203–0.369]

The adaptive variance threshold obtained a higher F1-score and a lower FPR than the agent, and the entropy detector obtained a similar F1-score with a much higher FPR. The agent therefore does not outperform a calibrated statistical detector in this scenario. Its advantage is that it learns from its reward without labeled training sessions, while the detectors needed labeled sessions to set their parameters, and that it also decides when to enable INT and when to block. The comparison has limits. The detectors were replayed on logs collected while the agent was acting, so the traffic they observed was shaped by its blocking, and they were not run in the control loop. Their parameters were tuned with labeled sessions, which favors them, and only one attack pattern was evaluated. The replay script is available in the repository [26].

4.2. Control Channel Interruptions

The controller reaches each switch through the Thrift interface, and every register read or rule installation is a separate call with a 15 s timeout. If the sketch cannot be read completely, the controller skips that window: it extracts no features, takes no decision, and leaves the INT mode and any installed drop rule unchanged. Installed rules remain in the switch tables, and the sketch keeps counting in the data plane. However, a failed call may leave the registers uncleared, so the window that follows an interruption can include packets from the skipped one. In the experiments, the controller and the switches ran on the same host and the attack traffic did not use the Thrift channel, so interruptions of this channel were not tested. The behavior of the agent under such interruptions remains to be evaluated.

4.3. Communication Delay and False Data Injection

The evaluation focuses on DoS attacks, but communication delays and false data injection (FDI) are also relevant in real deployments. In networked UAV control, delays degrade tracking performance [27], and FDI is addressed with controllers that compensate its effects on the measured data [28]. As noted above, the controller and the switches shared a host, so the delay of the control channel was not characterized either. The skip of incomplete windows covers failed reads, but a late read that still succeeds would change the span of the window, since the sketch registers accumulate packets until they are read and reset, and the features are packet counts. Regarding FDI, the agent assumes trusted switches and a trusted control channel. Its four features come only from the sketch registers, which the controller reads without authentication or integrity checks, and the reward is computed from the same signal. False data in these registers would therefore corrupt the state and the reward at the same time. Neither delay nor FDI was evaluated, and these statements come from an analysis of the design. Both remain open for future work.

4.4. Other Low-Rate Attack Variants

The experiments cover only the Shrew pattern, which is made of short and periodic bursts. Other low-rate attacks follow different patterns. Slowloris and slow HTTP POST, for example, use little bandwidth, keep connections open with incomplete or very slow requests, and closely resemble legitimate traffic [29,30]. Whether the agent adapts to them was not tested. The answer depends on what the agent observes and what it is rewarded for. As in any reinforcement learning agent, the state and the reward are defined by the designer, and the update rule only learns which action fits each state. Here, the state is built from the packet counts per source, destination, and protocol recorded in the sketch, and the reward identifies an attack when `max_est` exceeds 25 (Table 6). A slow attack would therefore influence the agent only if it raised those counts, which depends on the number of connections kept open and was not measured. The discretization thresholds

were also calibrated with the default parameters of Shrew. If adjustments were required, the main change would lie in the design of the state. With the current state, the agent cannot distinguish what the state does not represent, and no reward can compensate for that. The reward would also need a definition of attack that does not rely only on the threshold on `max_est`, since both come from the same packet counts. Attributes used in the literature for these attacks, such as packet length, inter-arrival time, flow duration, and connection-level statistics [29,30], are a possible starting point, but they were not implemented or tested. Adding features would enlarge the table of 5040 states. These statements come from an analysis of the design and not from experimental results. The extension to other low-rate variants is under development.

5. Conclusions

The results presented in Section 4 indicate that a Q-learning agent can distinguish LDoS attack traffic from legitimate traffic during network operation, without requiring prior labeled examples. Adaptive telemetry (Section 3.2) served as the agent's sole information source and maintained a stable average overhead of 5.22% on the control plane. The agent (Section 3.3) achieved a mean F1-score of 0.855 and detected 94.4% of attack episodes, with minimal impact on legitimate traffic in most sessions. An offline replay on the same logs, however, showed that a calibrated adaptive variance threshold reached a higher F1-score and a lower FPR than the agent. The contribution of the agent therefore lies in learning online without labeled training sessions and in controlling telemetry and mitigation together.

Two limitations were observed. A finer discretization of $\Delta_{\max}$ reduced the F1-score to 0.611, which indicates a granularity limitation of the tabular approach, and the average mitigation latency exceeded the baseline threshold, which is attributable to the hysteresis design. Work is under way to evaluate a DQN-based agent across a broader range of low-rate attack patterns. Future work includes physical P4 hardware, cooperative settings in which several agents share information, and the evaluation of the agent under communication delays and false data injection. Running statistical detectors inside the control loop and quantifying the trade-off between accuracy and cost against supervised and other reinforcement learning baselines in the same scenario also remain open.

The source code, the per-decision log of the evaluated sessions, and the analysis scripts are publicly available at [26]. Running the analysis script on the released log reproduces Table 7, with the two exceptions described in the Data Availability Statement.

Author Contributions: Conceptualization, A.V. and C.T.; methodology, A.V. and C.T.; software, A.V.; validation, A.V. and C.T.; formal analysis, A.V. and C.T.; investigation, A.V. and C.T.; resources, A.V. and C.T.; data curation, A.V.; writing—original draft preparation, C.T. and A.V.; review and editing, C.T. and A.V.; visualization, A.V.; supervision, C.T. and A.V.; project administration, C.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The code, logs, and analysis scripts supporting this study are available in a single repository [26] at https://github.com/Alizon-Valdiviezo/adaptive-ldos-detector (accessed on 22 October 2026) (release v1.1). The repository includes the P4 program, the controller, the traffic generators, the logs of the 21 sessions reported here with a mapping between the reported and original session numbers, the analysis scripts that produce the results of Section 4, and the content of the earlier degree project repository [6], which is archived. Running `python3 evaluacion_final.py results/decisions.jsonl 13` on the released log reproduces the values of Table 7, with two exceptions: the script reports attack episodes over all 21 sessions (133 episodes), whereas the table counts only sessions 13 to 21 (54 episodes), and throughput is computed from the per-session QoS logs in the repository. Running `python3 analysis/replay.py` on the released logs reproduces Table 8.

The agent learns online without a dataset, so there are no separate training data. No random seed was fixed, so repeating the experiment gives comparable but not identical results.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kuzmanovic, A.; Knightly, E. Low-rate TCP-targeted denial of service attacks and counter strategies. *IEEE/ACM Trans. Netw.* **2006**, *14*, 683–696. [CrossRef]
2. Xia, W.; Wen, Y.; Foh, C.H.; Niyato, D.; Xie, H. A Survey on Software-Defined Networking. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 27–51. [CrossRef]
3. Gutiérrez, S.; Botero, J.; Fletscher, L.; Gaspary, L.; Gaviria, N.; Jacob, E.; Lara, A.; Pérez, J.; To, M. Cybersecurity Via Programmable Networks: Are We There Yet? *J. Netw. Syst. Manag.* **2025**, *33*, 95. [CrossRef]
4. Yungaicela-Naula, N.M.; Vargas-Rosales, C.; Pérez-Díaz, J.A.; Carrera, D.F. A flexible SDN-based framework for slow-rate DDoS attack mitigation by using deep reinforcement learning. *J. Netw. Comput. Appl.* **2022**, *205*, 103444. [CrossRef]
5. de Almeida, L.C.; da Silva, W.R.D.; Tavares, T.C.; Pasquini, R.; Papagianni, C.; Verdi, F.L. DESiRED–Dynamic, Enhanced, and Smart iRED: A P4-AQM with Deep Reinforcement Learning and In-band Network Telemetry. *Comput. Netw.* **2024**, *244*, 110326. [CrossRef]
6. Valdiviezo, A. Hacia Redes Autónomas: Reinforcement Learning en SDN, Digital Twin Networks y Percepción 3D con Teledetección: Aprendizaje por Refuerzo en Entornos de Redes Definidas por Software. Undergraduate Thesis, Escuela Politécnica Nacional, Quito, Ecuador, 2026. (In Spanish) Available online: https://bibdigital.epn.edu.ec/handle/15000/27782 (accessed on 22 October 2026).
7. Wang, H.; Li, Y. Overview of DDoS Attack Detection in Software-Defined Networks. *IEEE Access* **2024**, *12*, 38351–38381. [CrossRef]
8. Tang, D.; Dai, R.; Yan, Y.; Li, K.; Liang, W.; Qin, Z. When SDN Meets Low-rate Threats: A Survey of Attacks and Countermeasures in Programmable Networks. *ACM Comput. Surv.* **2024**, *57*, 103. [CrossRef]
9. Alashhab, A.A.; Zahid, M.S.M.; Azim, M.A.; Daha, M.Y.; Isyaku, B.; Ali, S. A Survey of Low Rate DDoS Detection Techniques Based on Machine Learning in Software-Defined Networks. *Symmetry* **2022**, *14*, 1563. [CrossRef]
10. Clemens, V.; Schulz, L.C.; Gartner, M.; Hausheer, D. DDoS Detection in P4 Using HYPERLOGLOG and COUNTMIN Sketches. In Proceedings of the NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium, Miami, FL, USA, 8–12 May 2023; pp. 1–6. [CrossRef]
11. Sadi, A.A.; Savi, M.; Berardi, D.; Melis, A.; Prandini, M.; Callegati, F. Real-time Pipeline Reconfiguration of P4 Programmable Switches to Efficiently Detect and Mitigate DDoS Attacks. In Proceedings of the 2023 26th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 6–9 March 2023; pp. 21–23. [CrossRef]
12. Gori, G.; Rinieri, L.; Al Sadi, A.; Melis, A.; Callegati, F.; Prandini, M. GRAPH4: A Security Monitoring Architecture Based on Data Plane Anomaly Detection Metrics Calculated over Attack Graphs. *Future Internet* **2023**, *15*, 368. [CrossRef]
13. Khedr, W.I.; Gouda, A.E.; Mohamed, E.R. P4-HLDMC: A Novel Framework for DDoS and ARP Attack Detection and Mitigation in SD-IoT Networks Using Machine Learning, Stateful P4, and Distributed Multi-Controller Architecture. *Mathematics* **2023**, *11*, 3552. [CrossRef]
14. Musumeci, F.; Ionata, V.; Paolucci, F.; Cugini, F.; Tornatore, M. Machine-learning-assisted DDoS attack detection with P4 language. In Proceedings of the ICC 2020—2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–6. [CrossRef]
15. Zhang, H.; Chen, J.; Peng, H.; Chen, J. MAT4PM: Machine Learning-Guided Adaptive Threshold Control for P4-Based Monitoring in SDNs. *IEEE Trans. Netw. Serv. Manag.* **2026**, *23*, 3691–3706. [CrossRef]
16. Alashhab, A.A.; Zahid, M.S.M.; Abdullahi, M.; Rahman, M.S. Real-Time Detection of Low-Rate DDoS Attacks in SDN-Based Networks Using Online Machine Learning Model. In Proceedings of the 2023 7th Cyber Security in Networking Conference (CSNet), Montreal, QC, Canada, 16–18 October 2023; pp. 95–101. [CrossRef]
17. Liu, Y.; Dong, M.; Ota, K.; Li, J.; Wu, J. Deep Reinforcement Learning based Smart Mitigation of DDoS Flooding in Software-Defined Networks. In Proceedings of the 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Barcelona, Spain, 17–19 September 2018; pp. 1–6. [CrossRef]
18. Khozam, S.; Blanc, G.; Tixeuil, S.; Totel, E. DDoS Mitigation while Preserving QoS: A Deep Reinforcement Learning-Based Approach. In Proceedings of the 2024 IEEE 10th International Conference on Network Softwarization (NetSoft), Saint Louis, MS, USA, 24–28 June 2024; pp. 369–374. [CrossRef]
19. Shi, F.; Zhou, Z.; Guo, J.; Li, R.; Zhang, Z.; Li, S.; Liu, Q.; Bao, X. LightRL-AD: A Lightweight Online Reinforcement Learning Approach for Autonomous Defense against Network Attacks. In Proceedings of the 2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Sanya, China, 17–21 December 2024; pp. 1614–1621. [CrossRef]

20. Janabi, A.H.; Kanakis, T.; Johnson, M. Survey: Intrusion Detection System in Software-Defined Networking. *IEEE Access* **2024**, *12*, 164097–164120. [CrossRef]
21. Hajtmanek, R.; Kontšek, M.; Smieško, J.; Uramová, J. One-Parameter Statistical Methods to Recognize DDoS Attacks. *Symmetry* **2022**, *14*, 2388. [CrossRef]
22. Aladaileh, M.A.; Anbar, M.; Hintaw, A.J.; Hasbullah, I.H.; Bahashwan, A.A.; Al-Sarawi, S. Renyi Joint Entropy-Based Dynamic Threshold Approach to Detect DDoS Attacks against SDN Controller with Various Traffic Rates. *Appl. Sci.* **2022**, *12*, 6127. [CrossRef]
23. Cormode, G.; Muthukrishnan, S. An improved data stream summary: The Count-Min sketch and its applications. *J. Algorithms* **2005**, *55*, 58–75. [CrossRef]
24. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; The MIT Press: Cambridge, MA, USA, 2018; pp. 27–88.
25. Ibrahim, S.; Mostafa, M.; Jnadi, A.; Salloum, H.; Osinenko, P. Comprehensive Overview of Reward Engineering and Shaping in Advancing Reinforcement Learning Applications. *IEEE Access* **2024**, *12*, 175473–175500. [CrossRef]
26. Valdiviezo, A. Adaptive LDoS Detector: Autonomous Detection and Mitigation in P4-based SDN via Online Q-Learning. GitHub Repository, Release v1.1. 2026. Available online: https://github.com/Alizon-Valdiviezo/adaptive-ldos-detector (accessed on 22 October 2026).
27. Meng, Y.; Liang, H.; Liu, Z.; Huang, T. Composite Information Element-Based Self-Triggered Control for Distributed UAV Systems with Asymmetric Time-Varying Communication Delays. *IEEE Trans. Consum. Electron.* **2026**, *early access* . [CrossRef]
28. Zhang, H.; Yang, W.; Pan, Y.; Du, P. Prescribed-time resilient attitude control of multi-UAVs under unknown FDI attacks via dynamic memory event-triggered mechanism. *J. Frankl. Inst.* **2026**, *363*, 109055. [CrossRef]
29. Reed, A.; Dooley, L.S.; Kouadri Mostefaoui, S. Minimal Overhead Modelling of Slow DoS Attack Detection for Resource-Constrained IoT Networks. *Future Internet* **2025**, *17*, 432. [CrossRef]
30. Alrashid, B.H.; Alwadi, M.; Al-Haija, Q.A. Hybrid-Pipeline-Based Detection and Classification of HTTP Slow Denial-of-Service Attacks Using Radial Basis Function Neural Networks. *J. Cybersecur. Priv.* **2026**, *6*, 64. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.