Q-Learning-Based Detection of IPv6 Intrusions: A Behavioral and Performance Study

2025 International Seminar on Application for Technology of Information and Communication (iSemantic) | 979-8-3315-6591-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/ISemantic67418.2025.11291642

April Firman Daru
Faculty of Information Technology and Communication
Universitas Semarang
Semarang, Indonesia
firman@usm.ac.id

Alauddin Maulana Hirzan
Faculty of Information Technology and Communication
Universitas Semarang
Semarang, Indonesia
maulanahirzan@usm.ac.id

Zainab Senan Mahmod Attar Bashi
Kulliyyah of Information and Communication Technology
International Islamic University Malaysia
Selangor, Malaysia
zainab48@hotmail.com

Fajriannoor Fanani
Faculty of Information Technology and Communication
Universitas Semarang
Semarang, Indonesia
fajrian@usm.ac.id

Intrusion attacks remain a persistent threat in computer networks, occurring unpredictably across various geographic locations. Among these, IPv6-based flood attacks are particularly concerning due to the expanded address space, which enables the transmission of large packets. This problem can significantly affect local network performance or completely deny service to targeted servers. While numerous studies have proposed intrusion detection systems based on supervised learning models, a critical limitation persists. These models require retraining to recognise new attacks. This time-consuming retraining process may increase vulnerability during adaptation periods, as the networks will be exposed to zero-day attacks until updated training data is available. To address this limitation, the present study proposes a self-learning model using reinforcement learning techniques, specifically the Q-Learning algorithm, to classify network intrusions based on learned behavioural patterns autonomously. The system improves classification accuracy with each training epoch, enhancing its reliability. Three agents were designed, each employing different exploration-exploitation strategies characterised by epsilon E-0.1, E-0.5, and E-0.9. This study launched different ICMPv6 attacks individually and gathered five million samples for each intrusion attack. The agent with E-0.1 demonstrated superior performance, achieving 198,235 correct classifications with a cumulative reward of 883,835. The agent followed this with E-0.5, which recorded 100,984 correct classifications and a total reward of 87,075. The agent with E-0.9 performed the poorest, with only 20,850 correct classifications and a negative cumulative reward of -714,035. The findings indicate that the proposed self-learning model based on Q-Learning can effectively identify network intrusions without requiring manual retraining, thereby offering a scalable and adaptive solution for real-time intrusion detection.

Keywords—Classification, Intrusion, Patterns, Q Learning, Reinforcement Learning

I. INTRODUCTION

Network intrusions, particularly flood-based attacks, overwhelm systems with excessive traffic, rendering them inaccessible. They are different from other attack vectors like hacking or cracking. Flood attacks exploit packet transmission to overwhelm targets. There are many types of intrusions, but there is only one type of attack that is easy to use by many users. Flood-based intrusion is the easiest way to attack a network or computers actively. Unlike anomaly-based intrusion, flood-based intrusion sends a large number of packets to the target and fills up the bandwidth limit of that network. This attack effectively slows down or takes down a network or computers. Furthermore, IPv6 addressing technology increases the danger and risk of flood-based attacks. Unlike IPv4, IPv6 has a larger packet size. This statement means IPv6 carries more data than IPv4, making the flood attack more effective in taking down the targets. According to IPv4[1] and IPv6[2] Standards, the header size for IPv6 is twice as large as for IPv4. IPv6 has 320 bits (or 40 bytes) for the header. Meanwhile, IPv4 only has 160 bit (or 20 bytes) minimum size. What makes IPv6 intrusion more dangerous than IPv4 is the Jumbo Payload feature provided by IPv6. Theoretically, IPv6 with Jumbo Payload can hold 2^32 bytes, twice as large as the normal payload size in IPv4.

Many detection models have been proposed to mitigate the danger from IPv6 flood-based intrusions. The first study, in 2021, proposed a wireless network sensor to detect IPv6 using the Internet of Things node. This IoT model can detect and report flood-based intrusions through an MQTT protocol[3]. Within the same year, another study designed anomaly-based detection for the Neighbor Discovery Protocol. Instead of detecting anomaly behavior in standard protocols like ICMPv6, the study focused more on the NDP protocol. The designed model in the study used multiple algorithms like Entrophy-based Algorithm, Adaptive Threshold Algorithm, and Rule-based Algorithm[4]. The development of IPv6 flood detection continued in the following year. In 2022, a study proposed 6FloodDetector to detect intrusion based on flooding behavior through eigenvalues and threshold calculations[5]. The following article in the same year proposed a defense mechanism against IPv6 flood attacks (especially ICMPv6) using the P4-NSAF scheme. This scheme used Count-Min Sketch to defend against the attack[6]. In the following year, 2023, the article proposed IPv6 intrusion using traditional methods like supervised learning. Instead, the study designed a Reinforcement

Learning agent to classify whether a packet is an intrusion. Based on the evaluation, the designed agent accurately identified the intrusions and reached 98%. This accuracy was better than neural network algorithm[7]. The implementation of Machine Learning to detect intrusions was proposed in 2024. Unlike the traditional approach, which used rule-based detection, the machine learning-based approach created a model trained to detect intrusions. Thus, the model in the article[8] is capable of detecting ICMPv6-based DDOS. Meanwhile, the most recent study for IPv6 intrusion detection designed a combination of LSTM and GRU to detect ICMPv6 flood attacks. The designed model used a Long Short-Term Memory model trained with chosen features. Based on evaluation, the designed model has faster and more accurate detections[9].

Based on articles in the previous explanation, this study found each model's strengths and weaknesses points. Existing models lack dynamic adaptation, requiring retraining for new attack patterns, which delays response times. All models can only choose whether the traffic is an intrusion without the capability to classify what intrusion it is. Classifying intrusions is important to determine the level of danger. Hence, the administrator should know how to mitigate each danger level. Unable to determine the danger level of each intrusion carries a higher risk to the victims. The second weakness exists in supervised learning-based models, as in the article[7], [8], [9]. Besides that, the model proposed by the articles[8], [9] has a specific weakness. Since it is based on supervised learning, it cannot learn independently. The models need retraining to recognize new intrusions. This becomes a problem when the number of new intrusions is high. The training process would take longer and waste computational resources[10], [11], [12]. According to previous explanations, this study underlines the most important problems that exist within previous models: the incapability to self-learn without human interference and classification based on intrusion types[13].

To mitigate the weaknesses in the previous models, this study has the purpose of designing self-learning intrusion detection for ICMPv6-based floods using the Q Learn algorithm. Unlike previous studies, this study uses Reinforcement Learning as a detection mechanism capable of learning intrusion patterns. Besides that, the RL-based agent utilizes a memory table to learn the pattern as a State and decide on action according to the Q Value in each state. Hence, this study creates gaps in the current state-of-the-art. First, prior works train on limited attack patterns, increasing false negatives. Our model incorporates all known ICMPv6 flood variants to improve robust detection. Most models trained with ICMPv6-based intrusions but did not include the whole pattern of the intrusions. Thus, this problem increased the chance for the intrusion to slip by the detection system. Ensuring the model to learn all intrusion patterns is critical; missing one or more patterns increases the danger for the victims. The second gap is the methodological gap in several previous articles. Articles[8], [9] use a supervised learning approach to do intrusion detection in IPv6. This kind of approach is common to use since it is easy to design and implement. The last gap is the practical gap in article[7]. The article mentioned implementing Q Learn to decide whether IPv6 is an instruction. However, the capability to classify the intrusion type is not yet implemented. Thus, the designed Q Learn agent cannot determine each intrusion's danger level. Because of those reasons, this study proposes two contributions: a new approach to classifying IPv6 intrusions using off-policy Reinforcement Learning via the Q-learning algorithm and utilizing all patterns available in ICMPv6-based flood attacks as the dataset.

This study divides that article into several sections. The first section is the introduction, where this study explains the problem's background, past solutions, weaknesses in part solutions, and this study's proposed solution. The methodology explains what this study does to ensure the detection process's reproducibility with Q Learning. The results and discussions explain the result of this study model, including the discussion of its implications. The last section is the Conclusion, which summarizes from top to bottom of this article.

II. METHODOLOGY

This section explains how this study gathers the data, designs, evaluates, and validates the Reinforcement Learning (RL) Agent. The first explanation concerns how the study gathers IPv6 intrusion packet traffic through simulation, including how many samples were gathered. The second explanation is how this study designed the agent based on IPv6 characteristics and the agent's purpose. After the design phase, this study then explains the evaluation

A. Gathering The Data

Gathering the required data is an essential process before creating any RL agent. This study achieved that by using two virtual machines connected through a switch. Both machines are assigned IPv6 addresses to ensure they do not communicate with IPv4. After that, one machine is assigned as the attacker with THC-IPv6[14] installed and the other as the victim with Wireshark[15] installed. The next step is to choose what kind of attacks which categorized as ICMPv6-based attacks. After testing most of all available attacks in THC-IPv6, this study decided on 12 attacks that were categorized as ICMPv6 attacks. Table 1 contains the list of used attacks in this study.

TABLE I. INTRUSION TYPE

Attack Name	Description
Denial6 case 1	Denial of Service attack with Large Hop
Denial6 case 2	Denial of Service attack with Large Destination Header
Denial6 case 4	Denial of Service attack with Hop-by-Hop
Denial6 case 5	Denial of Service attack with AH Header
Flood_mld26	MLDv2 Reports floods
Flood_redir6	Router Advertisement flood
Flood_rs6	Router Solicitation flood
Flood_solicitate6	Neighbor Solicitation flood
Flood_unreach6	ICMPv6 Unreachable flood
Ndpexhaust26	ICMPv6 TooBig flood
Rsmurf6	ICMPv6 Echo Request Flood
Sendpees6	Neighbor Solicitation with CGA and RSA Verification

After deciding the intrusion type, this study launched each attack individually and gathered five million samples for each intrusion attack. This vast number of samples ensures that all patterns available in those attacks are properly gathered. Each sample is stored in packet capture new generation (pcapng) format before the next preprocessing step is completed.

B. Data Preprocessing

The next step is to preprocess the data before training them to an RL agent. This is an important step since each intrusion might have an empty or null value in each column.

The first step to preprocessing the data is to decide what column to take. This step ensures consistency across the intrusion when the training process runs. This study decided on 11 columns based on Wireshark filters and IPv6 patterns[16] as the primary feature for all intrusions. Table 2 explains the column used to display the intrusion characteristic.

TABLE II. INTRUSION FEATURES

Feature Name	Description
frame.len	Frame length
eth.type	Ethernet Type
ipv6.plen	IPv6 Total Length
ipv6.nxt	IPv6 Next Header
icmpv6.type	ICMPv6 Type
icmpv6.code	ICMPv6 Code
icmpv6.checksum	ICMPv6 Checksum
icmpv6.echo.identifier	ICMPv6 Echo Identifier
icmpv6.echo.sequence_number	ICMPv6 Echo Sequence Number
data.len	Data Length
data.data	Data

The characteristic combination in Table 2 shows the uniqueness of every intrusion. Thus, the cleaning process can be done to remove duplicates and extract the patterns. After removing the duplicates in the samples, this study obtained all unique patterns in every intrusion. Table 3 shows the available pattern in every intrusion.

TABLE III. INTRUSION PATTERN NUMBERS

Attack Name	Number of Patterns
Denial6 case 1	1
Denial6 case 2	1
Denial6 case 4	1
Denial6 case 5	1
Flood_mld26	65,536
Flood_redir6	21,846
Flood_rs6	1
Flood_solicitate6	65,536
Flood_unreach6	1
Ndpexhaust26	65,541
Rsmurf6	1
Sendpees6	1

Every attack has a different pattern. Some only have one pattern, and others have many patterns. Based on Table 3, all patterns did not exceed the sample number. Thus, this study successfully captured all possible attack patterns. After removing all duplicate patterns and extracting only unique patterns, this study continued the preprocessing stage by imputing missing values and converting non-decimal values. For imputing missing values, this study used the 99999 number as a replacement. This decision converts the missing value into something unique and distinct from other values used in IPv6. Conversely, converting non-decimal values to decimal ensures that all values inside the dataset are formatted as decimal numbers to align with the Q-table's numerical state representation. This preprocessing step will make the RL agent learn the pattern more easily than the non-decimal one.

C. Designing Q-Learning Agent

After the dataset was ready, this study continued to design the Reinforcement Learning agent using the Q-Learning algorithm. Unlike the supervised-learning-based model, which requires training over epochs before prediction, Reinforcement Learning-based agents use different approaches to prediction[17]. Suppose a supervised-learning-based model must be trained before classifying intrusions. In that case, a reinforcement-based agent uses online learning, where the agent learns continuously. For every step in the learning process, the agent will create a decision based on policy and actively reflect on whether its decision is correct or not. Whether correct or not, decisions will be recorded and used every time in the subsequent epochs.

This study created three different agents with three different decision policies. Table 4 exhibits the proposed Q Learning agent configuration.

TABLE IV. Q LEARNING MODEL PARAMETER

Parameter	Value
Learning Rate	0.1
Discount Factor	0.9
Epsilon	0.1, 0.5, 0.9
Random Seed	137318, 191446, 186765, 693667, 831766
Epochs	50
Action Space	Discrete: 12 Actions

Table 4 contains the configuration for the Q Learning agent. The configuration for the RL-based agent was different from that of the SL-based model. While both algorithms utilised Learning Rate as one of the parameters, the RL-based agent used additional parameters, like Discount Rate and Epsilon policy. This study configured the agent to learn with 0.1 as the learning rate. Meanwhile, the Discount Rate for Q value calculation was 0.9. These parameters were the default and often used in many situations. Thus, changing these values is only required when the agent produces an unexpected result. This study used a discrete action space with 12 actions to accommodate all intrusion classifications. Besides that, this study also configured the random seed to ensure consistency, since this algorithm was sensitive to the random seed. The configured random seed numbers were generated from a random function in Python. While using a similar configuration, this study created three agents based on Epsilon configurations. The first agent was configured to utilize a Q-table (or Memory Table) more than a randomized decision with an epsilon value of 0.1 (or E-0.1). The second one was configured to be balanced between a Q-table and a random decision with an epsilon value of 0.5 (or E-0.5), and the last one preferred a random decision instead of a Q-table with an epsilon value of 0.9 (or E-0.9). To simplify the explanation, lower epsilon (E-0.1) prioritizes exploitation of known Q-values, while higher epsilon (E-0.9) favors exploration, trading short-term accuracy for long-term adaptability. This configuration, called Epsilon Greedy, controls how the agent exploits the existing memory or explores the available options[18], [19].

Like supervised learning, the Reinforcement Learning agents must learn in a configured time-space called Epoch. Longer Epoch allows the model (in supervised learning) to converge its learning process properly. However, this statement did not fully apply to Reinforcement Learning agents. This study fixed the limit for learning epochs at 50 times. Anything beyond 50 times was considered resource waste since the agent converged its learning process below 50 times. Since a Reinforcement Learning agent requires a reward mechanism to boost memory, this study used +5 for correct decisions and -5 for incorrect decisions. This study can avoid bias toward correct or incorrect decisions using this balanced rewarding mechanism.

D. Evaluation

The last step is the evaluation; this study compared the learning performance between each epsilon configuration. This study compared three different results: Total Correct Actions, Total incorrect Actions, and Total Rewards. Total Correct Actions indicate how many times the agent made the correct decision. Total Incorrect Actions indicated the reverse. Meanwhile, Total Rewards reflected the Total Correct Actions calculation with the reward.

This study would not compare the agent with supervised learning (SL) models since the RL agent takes a different approach. There are several reasons why comparing supervised learning with reinforcement learning is similar to comparing apples to oranges. The first reason is that RL is an algorithm based on trial-and-error interactions in a configured environment, while SL-based models do not require this. The SL-based models relied on a static dataset, with ground truth included as the label. This algorithm also did not involve agent-to-environment interaction like the RL algorithm does[20]. The second reason is the objective of the algorithms. While SL-based models try to minimise loss functions, RL-based agents maximise long-term rewards[21].

Regarding evaluation complexity, SL-based models used straightforward metrics (like accuracy, precision) and were tied to ground truth from the dataset. Conversely, the RL-based agent used a different approach that utilised rewards as the base scoring[22], [23]. Thus, a comparison between SL-based models and RL-based agents as an evaluation was deemed unfair and improper.

III. Results and Discussions

In this section, this study explains the agent results and the comparison with different agent models. The first result is how the agent correctly classifying attack type based on their patterns. Figure 1 illustrates how E-0.1 performed against E-0.5 and E-0.9.

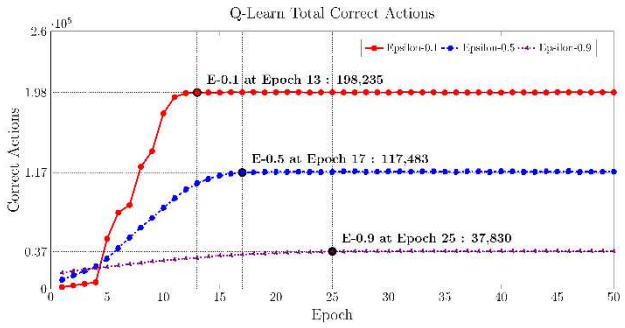

Fig 1. Total Correct Action Results between Agents

According to Figure 1, an agent with code E-0.1 reached the first highest correct classification at Epoch 13, averaging 198,235 actions. Followed by agent code E-0.5 with an average value of 117,483 correct actions at Epoch 17. The last agent code, E-0.9, stabilized at Epoch 25 with an average value of 37,830 correct actions. To add more evidence, this study calculates the incorrect classification and illustrates it in Figure 2.

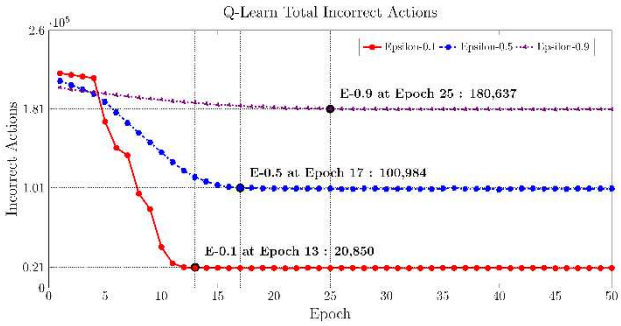

Fig 2. Total Incorrect Action Results between Agents

No matter how sophisticated a method is, it will not guarantee perfection. According to Figure 3, Agent E-0.1 suffered incorrect classifications, as many as 20,850 actions, and stabilized at Epoch 13. Agent E-0.5 suffered more incorrect decisions, as many as 100,984 at Epoch 17. Conversely, the highest number of incorrect decisions exists in Agent E-0.9, with 180,637 incorrect decisions at Epoch 25.

The last result is the Total Reward Calculated from the Total Correct Actions of each agent. Figure 3 shows that Agent E-0.1's reward (883,835) surpasses E-0.5 (87,075) and E-0.9 (-714,035), highlighting the trade-off between exploration (high epsilon) and exploitation (low epsilon).

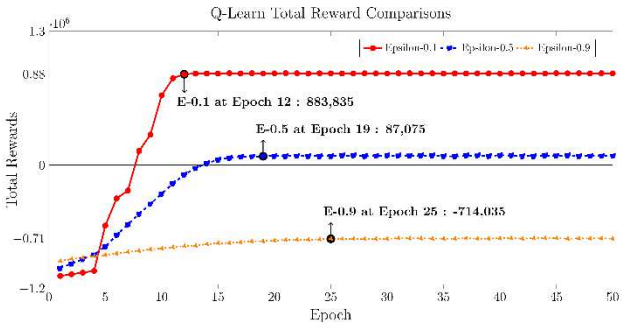

Fig 3. Total Reward Comparison between Agents

According to Figure 3, agent E-0.1 has the highest total rewards compared to agents E-0.5 and E-0.9. Agent E-0.1 reached 883,835 rewards, followed by Agent E-0.5 with 87,075 and Agent E-0.9 with -714,035. The total rewards obtained by each agent depend on how many of the agents decide on the intrusion correctly. Agent E-0.9 suffered negative rewards because the agent failed to determine the intrusion type.

This study explains the obtained validation results in this subsection. This study has been done on three validations: The first validation is to compare it with a baseline Q-Learning agent. Unlike the other agents in this study, the baseline agent was configured with an Epsilon of one and will randomly choose an action without any exploitation. Thus, this baseline agent is suitable to validate the proposed agent's performance. Figure 4 shows the performance comparison between the baseline and the proposed agents.

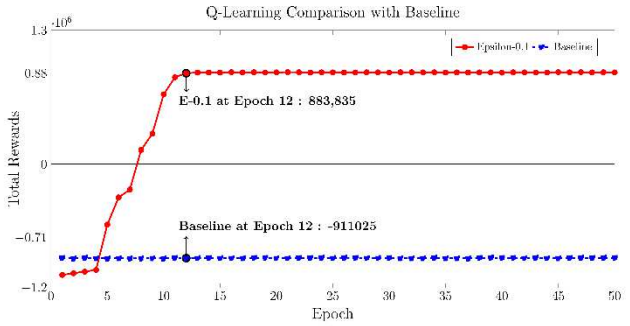

Fig 4. Total Reward Comparison between Agents

Based on Figure 4, the proposed agent with an Epsilon of 0.1 clearly showed reward differences. The baseline agent's reward only stayed around -911,025, as shown in the 12th epoch. Meanwhile, the proposed model reached a stable reward of 883,835. Figure 4 showed that Agent E-0.1 outperformed the baseline agent significantly.

The second validation is by creating a different E-0.1 that learns on different randomization seeds. Since the RL agent was affected by randomization caused by the Epsilon Greedy method, the produced result should be different. However, if the result is similar despite different random seeds, the RL agent can be concluded to have solid performance. Figure 5 illustrates the comparison between different seeds for the E-0.1 agent.

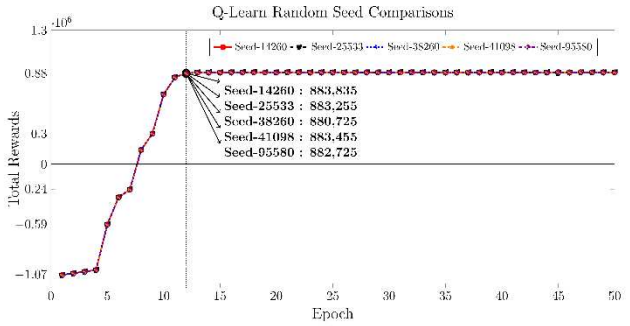

Fig 5. Total Reward Comparisons between Random Seeds

According to Figure 5, different random seeds barely affect the total rewards of Agent E-0.1. This study used five seeds; seed-14260 is the primary seed for the whole study process and has a total reward of 883,835. The next seed is seed-25533, with a total reward of 883,255. Seed-38260 has a total reward of 880,725, Seed-41098 has a total reward of 883,455, and the last seed, Seed-95580, has a total reward of 882,725. Despite differences in total rewards, all results are near each other and have small margin gaps. The subsequent validation is Action Entropy (known as Shannon Entropy) as the second validation with the following equation.

$$E(\cdot|s) = -\sum_{a} \pi(a|s) log \pi(a|s) \quad (1)$$

Where $E(\cdot|s)$ is the Action Entropy at state s or in this case is intrusion pattern. Meanwhile, $\pi(a|s)$ contains the probability of the selected action (as a) during classification of intrusion pattern (as s)[24], [25]. Figure 6 illustrates the result from Action Entropy between Agent E-0.1, E-0.5 and E-0.9.

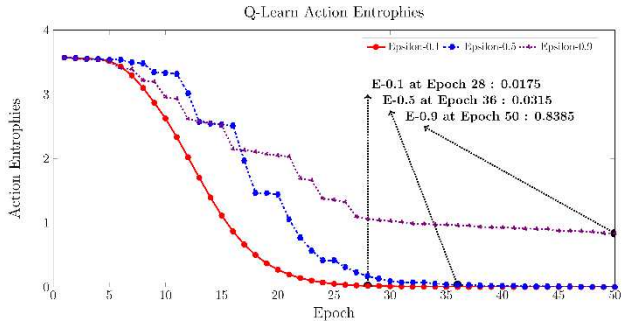

Fig 6. Action Entropy between Agents.

According to Figure 6, agent E-0.1 is the fastest agent, reaching the lowest point in Action Entropy at Epoch 28 with an Entropy of 0.0175. Agent E-0.5 started to stabilize at Epoch 36 with an Entropy of 0.0315. Meanwhile, agent E-0.9 has the highest Entropy with 0.8385 but has not stabilized at any Epoch. These results mean that agent E-0.1 has stabilised its policy and exploits its memory more than random action. It was followed by agent E-0.5, which was also successful in stabilising later on. However, agent E-0.9 never reached a stable Entropy at the end of an epoch. Since agent E-0.1 is configured with a more exploitative than explorative policy, the agent will choose what it has learned instead of a random action. Conversely, agent E-0.9 failed to stabilize because this agent preferred a random action over what it had learned in Q Table.

This subsection contains a discussion of the previous evaluation and validation results. The discussion subsection will explain what happened with the agents and their implications for real-life implementations. The first discussion concerns the results. The evaluation results in Figures 1 to 3 showed how reinforcement learning behaves when interacting with intrusions and their environments. The policy on each agent was the key finding of this reinforcement learning behavior study. Agent E-0.1 showed the fastest improvement at the 13th step compared to Agents E-0.5 (at the 17th step) and E-0.9 (at the 25th step) since the configured policy for E-0.1 was optimized for relying on the internal Q Table more instead of random action. The performance of Agent E-0.1 also exceeded the baseline model, where Agent E-0.1 reached the highest reward at 883,835 at the 12th step. This study used Random Seed comparison to validate these results since Reinforcement Learning is easily affected by randomization. Figure 5 shows Agent E-0.1 with a changed Random Seed, which has stable performance with slight changes. This result proved the stability of the Reinforcement Learning agent when faced with randomization changes. Besides that, Action Entropy validation was also calculated to show how fast the policy convergence on each agent occurred. The result showed that Agent E-0.1 was the fastest agent compared to Agents E-0.5 and E-0.9. Based on the evaluation and validation results, this study examines how well the proposed agent adapts to novel attack patterns, reducing vulnerability windows. A Reinforcement Learning agent is a self-evaluating, self-learning method that allows an agent to grow or learn during the training process.

The second discussion is about the comparison with previous studies. Compared with models in articles[5], [6], [8], [9], the proposed agent E-0.1 has a self-learning capability and can classify immediately without waiting for training time to finish. Conversely, this agent can better classify intrusion

types based on their pattern than the article[7]. The subsequent discussion is about the implications theoretically and practically. Reinforcement Learning is famous for its implementation in robotics, where this algorithm allows the robots to learn whether their decision or action is correct. However, this study proved that the RL algorithm implementations are not limited to that sector. The RL algorithm can be implemented in many sectors, including intrusion detection.

Furthermore, creating an intrusion detection agent based on the RL algorithm makes it possible to identify, classify, and decide on action (to allow, block, or drop) intrusion attacks. Meanwhile, on the practical side, the proposed intrusion classification agent can be used to identify and classify intrusion types before reaching the local network. Thus, this solution can mitigate any danger or risk before reaching the important data in the network. The subsequent discussion is about the strengths and weaknesses of the proposed agents. Based on the results, the proposed agent, especially E-0.1, performs better than other agents. It can learn faster and classify better because of its exploration and exploitation policies. However, there is a common weakness among the agents. Agents always start at zero. This means that every time the user runs the agents, they do not have memory from previous sessions. The agent needs time to learn before properly deciding on classification. Although it is possible to dump existing Q Table memory into a binary file (through the Python pickle library) and load it later, it would defeat the purpose of the self-learning mechanism in reinforcement learning. While cold starts delay initial deployment, future work could integrate transfer learning to bootstrap policies from past attacks.

The second weakness of this model is its implementation in a live network environment. Although it is possible to deploy this model to detect intrusions like NIDS usually does, preprocessing the traffic packet into a structured format, as this study did, is problematic. Most intrusion attacks forged by a third party did not have a consistent structure like a regular packet. Intrusions like NDPExhaust were padded with random data, resulting in a weird packet structure. Some intrusions have data, and the rest do not include any. Thus, preprocessing these malformed packets was challenging in a live environment.

The last weakness is the intrusion limit. The proposed agent in this study can only classify 12 different intrusions. Thus, the agent cannot classify more than that. However, this problem can be mitigated by adding a signature to help the agent identify the intrusion. The Q-table's fixed action space limits scalability, but modular architectures (e.g., hierarchical RL) could dynamically incorporate new attack classes. The last discussion is about future studies. The development of RL-based agents to classify intrusion will not stop here. Many areas can be improved; for example, adding a machine learning algorithm to improve action decisions better than the epsilon greedy method. According to discussions, this study concluded that the proposed agents, especially E-0.1, could classify intrusion with their self-learning mechanism.

IV. Conclusion

Intrusion attacks in networking are typical. They can happen at any time in any part of the world. However, ignoring these problems might increase the damage further. IPv6 is primarily based on flood attacks with larger packet space, which can slow down local network traffic or entirely block traffic access to the server. Because of that, many studies designed a supervised-based model to detect intrusions. However, there are many problems with the previous models, as they cannot learn on their own. They need to be retrained to detect new intrusions. The retraining process takes much time, and the damage from the intrusion will increase. With that weakness in the previous models, this study proposed self-learning agents capable of classifying intrusions based on their patterns. Every epoch passed, this agent will increase its confidence in classifying intrusions correctly. This study designed three agents with different exploration and exploitation policies. The first agent with configuration E-0.1 has the fastest and the best classification score compared to agents E-0.5 and E-0.9. This agent has correctly classified the intrusions 198,235 times with a total reward of 883,835, followed by agent E-0.5 with a correct classification of as many as 100,984 times with a total reward of 87,075. In last place is agent E-0.9, with the lowest correct classification of as many as 20,850, with a total reward of -714,035. This study then validated the results by creating different seeds for agent E-0.1. According to validation results, the performance did not have a significant margin in the total rewards. This study concluded that the designed model successfully classified the intrusion with a self-learning approach of the Q-Learning algorithm. Future work will address cold-start delays via transfer learning and expand attack coverage beyond 12 types. This approach bridges critical gaps in real-time, scalable intrusion detection.

Acknowledgment

The authors gratefully acknowledge the sponsorship and support provided through the international collaboration between the Kulliyyah of Information and Communication Technology, International Islamic University Malaysia, and the Faculty of Information and Communication Technology, Universitas Semarang.

References

[1] University of Southern California, *Internet Protocol*, Standards Track RFC 791, 1981. [Online]. Available: https://www.rfc-editor.org/rfc/rfc791

[2] S. Deering, CISCO, R. Hinden, and Nokia, *Internet Protocol, Version 6 (IPv6) Specification*, Standards Track RFC 2460, 1998. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2460#page-24

[3] A. Hendrawan, A. F. Daru, and A. M. Hirzan, "Intrusion detection with wireless sensor network (WSN) internet of things," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 13, no. 2, pp. 45–48, 2021.

[4] A. A. Bahashwan, M. Anbar, S. Manickam, I. H. Hasbullah, and M. A. Aladaileh, "Propose a Flow-Based Approach for Detecting Abnormal Behavior in Neighbor Discovery Protocol (NDP)," in *Advances in Cyber Security*, N. Abdullah, S. Manickam, and M. Anbar, Eds., Singapore: Springer Singapore, 2021, pp. 401–416.

[5] L. Zhang, W. Xia, W. Huang, W. Du, Y. Guo, and L. Cheng, "6FloodDetector: An IPv6 Flooding Behaviors Detection Technology Based on Eigenvalues and Thresholds," in *2022 IEEE 22nd International Conference on Communication Technology (ICCT)*, Nov. 2022, pp. 1375–1379. doi: 10.1109/ICCT56141.2022.10072952.

[6] Y. Li, W. Yang, Z. Zhou, Q. Liu, Z. Li, and S. Li, "P4-NSAF: defending IPv6 networks against ICMPv6 DoS and DDoS attacks with P4," in *ICC 2022 - IEEE International Conference on Communications*, May 2022, pp. 5005–5010. doi: 10.1109/ICC45855.2022.9839137.

[7] A. F. Daru, K. D. Hartomo, and H. D. Purnomo, "IPv6 flood attack detection based on epsilon greedy optimized Q learning in single board computer," *IJECE*, vol. 13, no. 5, pp. 5782–5791, 2023, doi: http://doi.org/10.11591/ijece.v13i5.pp5782-5791.

[8] A. E. Ksimi, C. Leghris, S. Lafraxo, and V. K. V. and, "ICMPv6-based DDoS Flooding-Attack Detection Using Machine and Deep Learning Techniques," *IETE Journal of Research*, vol. 70, no. 4, pp. 3753–3762, 2024, doi: 10.1080/03772063.2023.2208549.

[9] Y. Cheng and H. Xu, "Research on ICMPv6 DDoS Attack Detection Based on Integrated Feature Selection and LSTM-GRU," in *Proceedings of the 2024 2nd International Conference on Artificial Intelligence, Systems and Network Security*, in AISNS '24. New York, NY, USA: Association for Computing Machinery, 2025, pp. 227–233. doi: 10.1145/3714334.3714373.

[10] M. A. Umer, K. N. Junejo, M. T. Jilani, and A. P. Mathur, "Machine learning for intrusion detection in industrial control systems: Applications, challenges, and recommendations," *International Journal of Critical Infrastructure Protection*, vol. 38, p. 100516, Sep. 2022, doi: 10.1016/j.ijcip.2022.100516.

[11] C. Zhang, D. Jia, L. Wang, W. Wang, F. Liu, and A. Yang, "Comparative research on network intrusion detection methods based on machine learning," *Computers & Security*, vol. 121, p. 102861, Oct. 2022, doi: 10.1016/j.cose.2022.102861.

[12] T. Talaei Khoei and N. Kaabouch, "A Comparative Analysis of Supervised and Unsupervised Models for Detecting Attacks on the Intrusion Detection Systems," *Information*, vol. 14, no. 2, 2023, doi: 10.3390/info14020103.

[13] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions," *SN Computer Science*, vol. 2, no. 3, p. 160, Mar. 2021, doi: 10.1007/s42979-021-00592-x.

[14] Van Hauser, *The Hacker Choice's IPv6 Attack Toolkit*. (2020). C. [Online]. Available: https://github.com/vanhauser-thc/thc-ipv6/tree/v3.8

[15] Wireshark Foundation, *Wireshark*. (2025). C. [Online]. Available: https://gitlab.com/wireshark/wireshark/-/tree/release-4.2?ref_type=heads

[16] J. Davies, *Understanding ipv6*. Pearson Education, 2012.

[17] J. Clifton and E. Laber, "Q-Learning: Theory and Applications," *Annu. Rev. Stat. Appl.*, vol. 7, no. 1, pp. 279–301, Mar. 2020, doi: 10.1146/annurev-statistics-031219-041220.

[18] Q. Nguyen, N. Teku, and T. Bose, "Epsilon Greedy Strategy for Hyper Parameters Tuning of A Neural Network Equalizer," in *2021 12th International Symposium on Image and Signal Processing and Analysis (ISPA)*, Sep. 2021, pp. 209–212. doi: 10.1109/ISPA52656.2021.9552055.

[19] J. Li and I. O. Ryzhov, "Convergence Rates of Epsilon-Greedy Global Optimization Under Radial Basis Function Interpolation," *Stochastic Systems*, vol. 13, no. 1, pp. 59–92, Mar. 2023, doi: 10.1287/stsy.2022.0096.

[20] R. Verma, V. Nagar, and S. Mahapatra, "Introduction to Supervised Learning," in *Data Analytics in Bioinformatics*, 2021, pp. 1–34. doi: 10.1002/9781119785620.ch1.

[21] Z. Xu, B. Liu, X. Xiao, A. Nair, and P. Stone, "Benchmarking Reinforcement Learning Techniques for Autonomous Navigation." 2023. [Online]. Available: https://arxiv.org/abs/2210.04839

[22] K. K. Jha, R. Jha, A. K. Jha, M. A. M. Hassan, S. K. Yadav, and T. Mahesh, "A Brief Comparison On Machine Learning Algorithms Based On Various Applications: A Comprehensive Survey," in *2021 IEEE International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, Dec. 2021, pp. 1–5. doi: 10.1109/CSITSS54238.2021.9683524.

[23] J. Fan, C. Xiao, and Y. Huang, "GDI: Rethinking What Makes Reinforcement Learning Different From Supervised Learning." 2022. [Online]. Available: https://arxiv.org/abs/2106.06232

[24] P. M. Cincotta, C. M. Giordano, R. Alves Silva, and C. Beaugé, "The Shannon entropy: An efficient indicator of dynamical stability," *Physica D: Nonlinear Phenomena*, vol. 417, p. 132816, Mar. 2021, doi: 10.1016/j.physd.2020.132816.

[25] A. Ali, S. Anam, and M. M. Ahmed, "Shannon Entropy in Artificial Intelligence and Its Applications Based on Information Theory," *Journal of Applied and Emerging Sciences; Vol 13, No 1 (2023)*, 2023, [Online]. Available: https://journal.buitms.edu.pk/j/index.php/bj/article/view/549