International Journal of Electrical and Computer Engineering (IJECE)
Vol. 13, No. 5, October 2023, pp. 5782~5791
ISSN: 2088-8708, DOI: 10.11591/ijece.v13i5.pp5782-5791

IPv6 flood attack detection based on epsilon greedy optimized Q learning in single board computer

April Firman Daru[1], Kristoko Dwi Hartomo[2], Hindriyanto Dwi Purnomo[2]
[1]Faculty of Information Technology and Communication, Semarang University, Semarang, Indonesia
[2]Faculty of Information Technology, Satya Wacana Christian University, Salatiga, Indonesia

Article Info

Article history:

Received Sep 13, 2022
Revised Apr 18, 2023
Accepted Apr 24, 2023

Keywords:

Epsilon greedy
Intrusion detection
IPv6 flooding
Off policy Q learning
Reinforcement learning
Single board computer

ABSTRACT

Internet of things is a technology that allows communication between devices within a network. Since this technology depends on a network to communicate, the vulnerability of the exposed devices increased significantly. Furthermore, the use of internet protocol version 6 (IPv6) as the successor to internet protocol version 4 (IPv4) as a communication protocol constituted a significant problem for the network. Hence, this protocol was exploitable for flooding attacks in the IPv6 network. As a countermeasure against the flood, this study designed an IPv6 flood attack detection by using epsilon greedy optimized Q learning algorithm. According to the evaluation, the agent with epsilon 0.1 could reach 98% of accuracy and 11,550 rewards compared to the other agents. When compared to control models, the agent is also the most accurate compared to other algorithms followed by neural network (NN), K-nearest neighbors (KNN), decision tree (DT), naive Bayes (NB), and support vector machine (SVM). Besides that, the agent used more than 99% of a single central processing unit (CPU). Hence, the agent will not hinder internet of things (IoT) devices with multiple processors. Thus, we concluded that the proposed agent has high accuracy and feasibility in a single board computer (SBC).

Corresponding Author:

April Firman Daru
Faculty of Information Technology and Communication, Semarang University
Soekarno-Hatta Street, West Tlogosari, Semarang 50196, Indonesia
Email: firman@usm.ac.id

1. INTRODUCTION

Network intrusions are a part of global network connectivity that occur everywhere and may cause problems to all connected devices. From year to year, network intrusions are still growing on many sides such as attack patterns and protocols. Before internet protocol version 6 (IPv6) was introduced, internet protocol version 4 (IPv4) was often used as a medium to attack a network. But in a recent article, many hackers started targeting IPv6-connected devices with flooding attacks [1]. Similar to the IPv4 protocol, the IPv6 network protocol also has weaknesses in terms of security. This protocol could be used for flooding attacks with different levels of risk [2], [3].

There are many types of intrusion that exist, one of them is known as denial of service. This kind of intrusion floods spams data in massive numbers to shut down internet of things devices such as IP Cameras or other types of monitoring devices [4]. Since those devices have lower processing capability, it was easier for intrusions from outside to knock down the devices [5]–[7]. Another factor that increases the risk of the device is the unattended mechanism that allows devices to work without human interference [8], [9]. Thus, the security enhancement for Internet of Things devices became a trend and a challenge for future research [10].

Many articles of research focus on IPv6 mitigation on the internet of things (IoT). The development of IPv6 intrusion detection started from signatures-based until machine learning-based detection. Signature-based intrusion detection is easier to implement on the internet of things since high processing power is not required. The devices only need to read the rules and compare the characteristic of the data to determine the detection. This kind of detection has high accuracy since the model only needs to check the existing signature with the data [11], [12]. However, there is a problem detecting a complex attack. The signature-based detection is not well-suited for complex detection, so machine learning is invented to overcome the problem [13], [14]. Machine learning-based detection usually uses a feature classification to detect intrusion into the network [15]–[17]. For example, IPv6 intrusion detection for router advertisement flooding has an accuracy of up to 98.55%. This model can detect IPv6-based intrusion effectively with a machine learning algorithm [18]. Even though machine learning-based detection has high detection accuracy, the implementation on the Internet of Things device was not feasible due to its limitations [19]. Besides that, the supervised learning method in machine learning only guesses the correct answer based on the trained model. Hence to improve its accuracy, the researchers must be involved. Incapability to improve its accuracy without human interference is the main weakness of the supervised learning method in the previous articles of the research [20].

Because of this reason, this study tries a different approach to developing intrusion detection with epsilon greedy optimized Q learning. Unlike the supervised learning method, this method not only guesses the correct answer but also improves itself in the shape of reward feedback [20]. Q learning itself is a reinforcement learning method that makes an internet of things device an agent to learn the characteristics of the intrusions in the IPv6 network. Hence, the device can determine whether the data is an intrusion or neutral. This paper stated the contributions of the study with the following statements. We proposed a reinforcement learning-based flooding attack detection model based on the IPv6 package pattern. Unlike the current state-of-the-art model that cannot improve its accuracy, the proposed model use epsilon greedy optimized Q learning-based as the self-improving detection model.

2. THE PROPOSED METHOD

2.1. Data gathering

In this section, we explain the proposed method used to solve the problem that exists in the previous studies. The proposed method consists of several parts such as the data sample for training and testing, the algorithm, the environment of the agent, and the agent itself. To build an agent that is capable to detect IPv6-based attacks, this study gathered several intrusion characteristics by capturing live. The setup topology for data gathering is illustrated in the Figure 1.

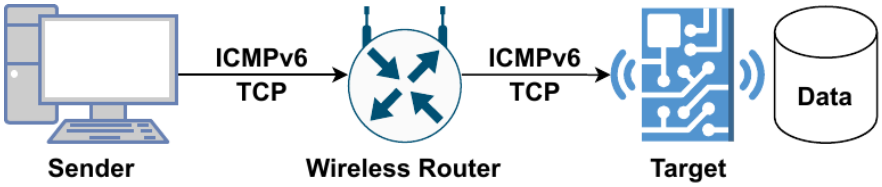

Figure 1. Simple IPv6 network topology to capture data

Figure 1 is the process for gathering the required data. To obtain both neutral and intrusion data, this study used two computers that connected via a wireless network to simulate the flooding attacks. One computer is given a role as an attacker and equipped with The Hacker's Choices tools (THC-IPv6) to flood the victim with IPv6-based data. The used tools in this experiment consist of *denial6*, *flood_unreach6*, *thcsyn6*, *nping*, and *fping* [21]. Meanwhile, the target computer is an internet of things processing board called Raspberry Pi that is equipped with Wireshark to collect flood data. This study gathered two types of data that consist of neutral and intrusion data. Each type also consists of two different protocols such as transmission control protocol (TCP) and internet control message protocol version 6 (ICMPv6). The tools used to gather TCP-based data such as thc-syn6 (as intrusion sample) and *nping* (as neutral sample). Meanwhile, *denial6* (as intrusion sample) and *fping* (as neutral sample) are used to gather ICMPv6-based data.

Table 1 explains the used flooding tools from the THC-IPv6 toolkit in the experiment. The attacks consist of two protocols, ICMPv6 and TCP where each protocol has five different toolsets used as the dataset generator. To generate the ICMPv6 dataset, we use *fping* to generate normal ICMPv6. Meanwhile, we use *denial6* from the THC-IPv6 toolkit with two different packet generation switches (one for hop-by-hop, and one for large unknown option); *flood-unreach6* for flooding the target with unreachable packets; *rsmurf6* to smurf

the target (as part of distributed denial-of-service act). For the TCP data, we use *nping* to generate normal TCP packets, thc-syn6 with four different switches to attack the target. The first switch for *thcsyn6* will generate TCP-synchronize (TCP-SYN) packets, the second switch generates TCP-acknowledge (TCP-ACK) packets, the third switch generates TCP-SYN-ACK packets, and the last one generates hop-by-hop router alert TCP packets.

Table 1. Attacks performed during data gathering

Num	Performed THC command	Protocol	Packet details
Atk 1	*fping*-6 [target]	ICMPv6	Neutral normal packet
Atk 2	*denial6* [interface] [target] 1	ICMPv6	Flood attack with large hop-by-hop header
Atk 3	*denial6* [interface] [target] 2	ICMPv6	Flood attack with large unknown option header
Atk 4	*flood-unreach6* [interface] [target]	ICMPv6	Flood target with unreachable ICMPv6 packets
Atk 5	*rsmurf6* [interface] [target]	ICMPv6	Smurf target in local network
Atk 6	*nping*-6–tcp [target]	TCP	Neutral normal packet
Atk 7	*thcsyn6* [interface] [target]	TCP	Flood victim with TCP-SYN
Atk 8	*thcsyn6*-A [interface] [target]	TCP	Flood victim with TCP-ACK
Atk 9	*thcsyn6*-S [interface] [target]	TCP	Flood victim with TCP-SYN-ACK
Atk 10	*thcsyn6*-a [interface] [target]	TCP	Flood victim with hop-by-hop router alert

However, the obtained data from the data-gathering phase contains many unneeded fields. Since not all fields are required, this study pre-processes the raw dataset into a finer dataset that contains required fields. In some cases, there is value similarity inside a dataset, this study decided to choose at least one field to be uniquely allowed in the dataset. Table 2 contains the used fields in the dataset:

Table 2. The packet characteristics for learning target

Feature	ICMPv6	TCP
1	Source *Addr*	Source *Addr*
2	Destination *Addr*	Destination *Addr*
3	Protocol	Protocol
4	Length	Length
5	Payload length	Payload length
6	Type	Window size
7	Data	Flags
8	Detection	Detection

According to Table 2, the data fields were taken from the header and the payload of the data. In the header part, there are, source address; destination address; protocol; length; and payload length are used as unique fields. The header fields between TCP and ICMPv6 are the same since both protocols already have unique values. The ICMPv6 payload part consists of type and data. Mean-while TCP payload part consists of window size and flags. The last field labelled detection contains a manually assigned Boolean value to indicate whether the data is an intrusion or not.

After completing the labelling process, this study continues the next portioning data process. This study decides on a fair 50:50 portion for both intrusion and neutral packets (not based on the tools used). Making this fair portioning will help prevent the agent to turn one side only. Besides data row size, this study also portioned the data rows according to the protocol and its data type. Table 3 explains the portioning of the data rows according to the protocol and its data type.

Table 3. Dataset sample and compositions

Protocol	Tool(s)	Row(s)	Type
ICMPv6	*fping*	1,000	Neutral
	denial6 case 1	250	Intrusion
	denial6 case 2	250	Intrusion
	flood-unreach6	250	Intrusion
	rsmurf6	250	Intrusion
TCP	*nping*	1,000	Neutral
	thcsyn6 no-opt	250	Intrusion
	thcsyn6 ACK	250	Intrusion
	thcsyn6 SYN-ACK	250	Intrusion
	thcsyn6 hop-by-hop	250	Intrusion
Total data rows		4,000	-

The ICMPv6 protocol has 1,000 rows of *fping* data, 250 rows of *denial6* test case1 data, 250 rows of denial 6 test case 2 data, 250 rows of *flood_unreach6* data, and 250 rows of *rsmurf6* data. Hence the ICMPv6 has 1,000 neutral and 1,000 intrusion data. Meanwhile, the TCP protocol has 1,000 rows of *nping* data, 250 rows of *thcsyn6* without option data, 250 rows of *thcsyn6* ACK data, 250 rows of thc-syn6 SYN-ACK data, and 250 rows of *thcsyn6* hop-by-hop data. Similar to the ICMPv6 protocol, TCP has 1,000 neutral and 1,000 intrusion data. Adding various intrusion types to the dataset will increase the agent's knowledge about the intrusions. All dataset that contains neutral and intrusion data is stored inside a CSV file for easier access. Before the training process starts, the agent load and split the dataset into 70:30 stratified training and test data. The stratifying process during data split has the purpose to balance the number of rows in each data field. At the end of dataset pre-processing, this study obtained 2,800 rows of training data and 1,200 rows of test data randomized.

2.2. Environment design

After the data pre-processing phase is complete, this study designed an environment for the agent to learn the data. However, the environment used for intrusion detection is different from publicly available environments. The problem lies in the numbering system that the environment uses. The publicly available environments used rational numbers as their states. Thus, the dataset is not compatible with the current environment. To solve this problem, this study changed the numbering system in the environment with the whole number system. Besides changing the number system, this study also used number conversion through truncated decimal converted SHA-1 checksum hash to change any values inside the data set into unique numbers. Figure 2 illustrated the process of the number conversion of the dataset.

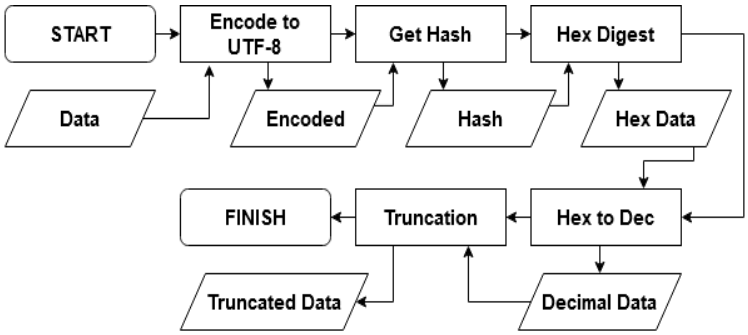

Figure 2. Data to decimal number conversion method

Figure 2 contains a method to convert any data type into unique numbers starting by encoding each data into a UTF-8 string. The next step is to get the hash result of the string with the SHA-1 algorithm and turn it into hexadecimal through digest. The decimal value can be obtained by the decimal conversion process of hexadecimal hash. However, the result of the conversion is too long for the agent to store. Hence, the result from the previous process is truncated into ten digits. This number is unique and useful to distinguish between intrusion A and B. By using this method, the environment will accept the truncated decimal data.

The next process is to configure the reward mechanism in the environment. The reward is a feedback mechanism that reinforcement learning uses to optimize the agent's decision mechanism. The calculation of the reward inside the environment uses IF-based rules by matching the detection indicator inside the data set with the action taken by the agent. From this point, the environment can raise four different detection indicators. Table 4 contains the reward calculation and detection indicators.

Table 4. Reward calculation and detection indicators

Indicator	Detection	Agent's action	Rewards
True positive (TP)	True	True	+10
True negative (TN)	False	False	+10
False positive (FP)	False	True	-5
False negative (FN)	True	False	-5

According to Table 4, the agent will receive positive rewards if the agent determines the correct action and value (true positive (TP) and true negative (TN)). In the study, the environment will return ten points if the agent determines correctly. Meanwhile, the agent will receive negative rewards if the agent determines the

values incorrectly (false positive (FP), false negative (FN)). The environment will return minus five points and decrease the accumulated rewards. The accumulated rewards are usable for action decision factors and performance evaluation at a later stage. The agent will determine the next action according to the accumulated rewards. Besides that, high reward accumulations mean that the detection has good accuracy.

2.3. Q learning agent

The last step of the environment design phase is to build the agent and its interaction with the environment. The agent is the main system that determines whether a packet is an intrusion or not in this study. However, the agent needs to interact with the environment to determine the correct action for each data row in the dataset. The interaction between agent and environment will produce a state and a reward. The process of the interaction for agent training and testing is illustrated in Figure 3.

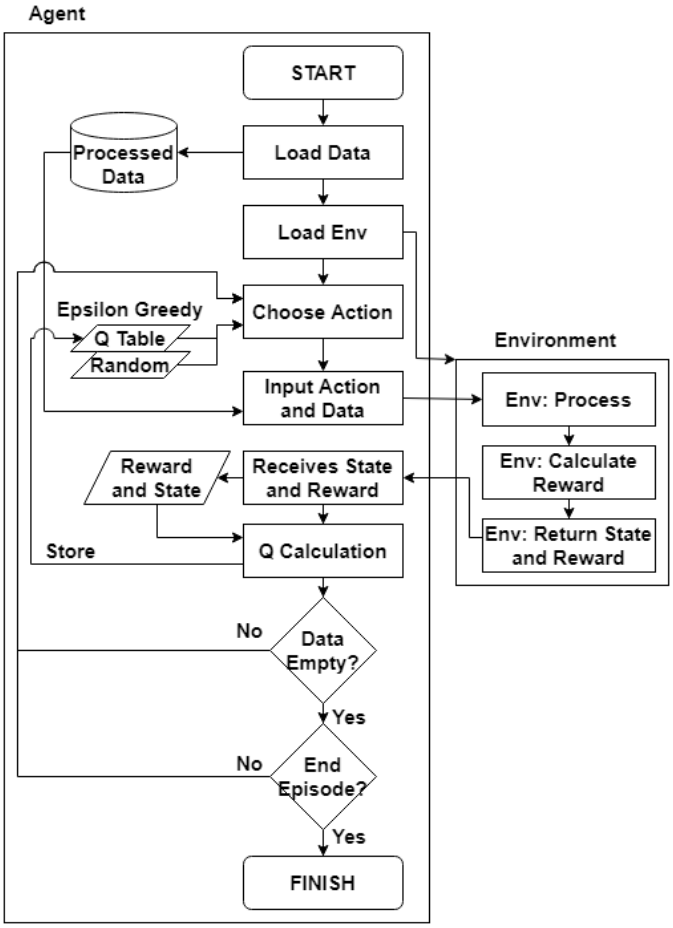

Figure 3. Agent algorithm with epsilon greedy and Q learning

According to Figure 3, the interaction starts with the agent loading pre-processed training data and the environment. After the loading process is complete, the agent chooses the action between random and Q Table with the epsilon greedy method assistance [22], [23]. The formula used for Epsilon greedy is shown (1):

$$A^t\begin{cases} max\ Q(A) with P(1-\epsilon) \\ RandomAction(A) with P(\epsilon) \end{cases} \quad (1)$$

where A is the action taken for the agent, Q is the Q Table, P is the probability of action taken, t is the time or step, and ϵ is the value of the probability. Since this method uses probability, the agent will receive an action from the maximum policy table or random action. Theoretically, forcing the agent to use the maximum policy inside the Q table more often can increase the accuracy. It means that the learning model needs to explore first and then exploit the result to achieve the best performance [24], [25]. At this point, the agent already has the

dataset and the action. The agent inputs a data row and action into the environment and let the environment calculate the reward. The environment returns the reward and the state after the process is complete. The agent receives the state and the reward and evaluates the learning process with the Q learning algorithm. As shown by (1) is the formula used for Q learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma.maxQ(S_{t+1}, a) - Q(S_t, A_t)] \quad (2)$$

This formula consists of several elements such as reward (*R*), state (*S*) dan action (*A*). Besides that, this formula also counts t as the time step or episode, α as the agent's learning rate, and γ as the reward discount rate. These variables are also known as hyperparameters that may affect the learning process if changed. This function is known as Q-Function or action-value function where the agent can determine the specific action to take when exploiting the Q Table. Since this algorithm is an off-policy algorithm, the agent cannot decide between exploration and exploitation explicitly. The agent stores the evaluation result in the coordinate of the Q Table with the state on X-axis and action on Y-axis. The agent will repeat these steps until the specified episodes in the training phase. Meanwhile, during the testing phase, the agent only needs to do it once with Q Table as the action source.

3. METHOD

This section explains the agent's performance evaluations by following four experiments with different scenarios. Each scenario has a different epsilon size to evaluate the performance of intrusion detection. With these scenarios, this study can understand the relationship between exploration and exploitation with detection results. Table 5 explains the experiment's scenarios.

Table 5. Experiment scenario setup for evaluation

Scenario	Epsilon	Time Epoch	Training	Test
1	0.1	10	10	1
2	0.5	10	10	1
3	0.9	10	10	1
4	1.0	10	10	1

According to Table 5, this study uses four different scenarios to evaluate the learning capability of the agent. Each scenario has a different epsilon configuration but the same training and test episodes. The epsilon configuration in the table starts from the best-policy action (0.1) to pure random action (1.0). Since the range is quite wide, this study only chooses the most significant epsilon value (0.1, 0.5, 0.9, 1.0). In the terms of training and testing epochs, this study will execute the training for ten episodes starting from one. This experiment uses a shorter period since the dataset contains repeated data and is sufficient to train the agent. Meanwhile, the testing epoch is only one episode. This phase force the agent to use the best action available in the table to test the accuracy of the detection. To obtain the accuracy of detection, this study uses a confusion matrix to populate the detection results. With the help of the confusion matrix, this study can calculate the accuracy of the detection. Thus, this study can understand better how the agent learns IPv6-based intrusions [26]. Hence, the equation for the accuracy is in (3),

$$Accuracy(\%) = \frac{(TP+TN)}{(TP+TN+FP+FN)} \quad (3)$$

where TP and TN. These two indicators are indicated that the agent chooses the correct action for the packet. The sum value of these two indicators is divided by the sum of all indicators (TP, TN, FP, and FN). The result of division is the accuracy of the detection. A higher value indicates that the agent has a good detection of the intrusions.

Besides the accuracy benchmark, this study also uses a reward graph to evaluate how well the agent performs. If the confusion matrix focuses on how good the detection is, then the reward graph shows how good the agent chooses the correct action for each data. This type of evaluation is not feasible in supervised learning since the algorithm does not use an agent to do the learning process. As the control for the accuracy evaluation, this study compares machine learning-based intrusion detection with the agent.

Using a reward graph as the evaluation aspect, this study can compare the agent's performance to pick the correct action for each data. If the agent chooses the correct action, then the accumulated rewards will increase. But if the agent incorrectly chooses the action, the accumulated rewards will decrease. Also, if the agent has maximum accumulated rewards, it means that it can correctly determine all the test data.

The last aspect of the evaluation is the processing performance of the internet of things device. Since this study uses Raspberry Pi as the target device, this study also needs to gather performance evidence during the whole process. Benchmarking the performance of the agent inside the Raspberry Pi device, this study can understand the impact of reinforcement learning inside an IoT device. In this aspect, the research can gather the CPU and the memory usage during the training and testing phases.

4. RESULTS AND DISCUSSION

This section elucidates the result of the agent's evaluation. The results contained a detection summary, agent's accuracy and reward graphs, accuracy comparison with different algorithms, and performance benchmark. The first evaluation is the detection results during experiments, the data stored in a shape of a table with TP, TN, FP, and FN indicators. The second evaluation is the agent's performance with accuracy and reward graph. The third evaluation was the comparison result between other classification algorithms. The last one was the hardware performance benchmark for epsilon greedy optimized Q learning for low-end devices like single board computer (SBC).

According to Table 6, agent 0.1 correctly determines the test data during the experiment. This agent did not have a value larger than 0 in false positive and false negative indicators. Unlike agent 0.1, the other agents did not have zero results in their results. This table showed that a higher epsilon value can lower the result in true indicators. Hence, the accuracy of the detections should be lower. To prove this statement, this study calculated Table 6 results into accuracy. Then, compare the accuracy and the rewards side by side for each agent with different epsilon. Figure 4 shows the comparison result between each agent.

Table 6. Q learning agent's average detection results

Agent	TP	TN	FP	FN
0.1	570	600	0	30
0.5	480	510	90	120
0.9	488	330	270	112
1.0	465	134	123	478

According to Figure 4, the agent with epsilon 0.1 has the highest accuracy and rewards. With average detection accuracy up to 98% and average rewards of 11,500, this agent outperformed the rest of the agents. Meanwhile, the result of each agent was: the agent epsilon 0.5 in the second place with the accuracy reached 83% and accumulated reward up to 8,850, in the third place is the agent epsilon 0.9 with accuracy reached 68% and reward of 6,262, and the last place is the agent without learning reached accuracy up to 50% and reward 2,974.

The next step for the evaluation is to compare with the control model from another machine learning algorithm. Using the published article as the main reference for comparison, this study put the result of the comparison side by side. The cited references were using a similar tool to generate the intrusions, but different in the terms of detection model. Figure 5 showed the comparison between this agent's accuracy with the model from the article [27].

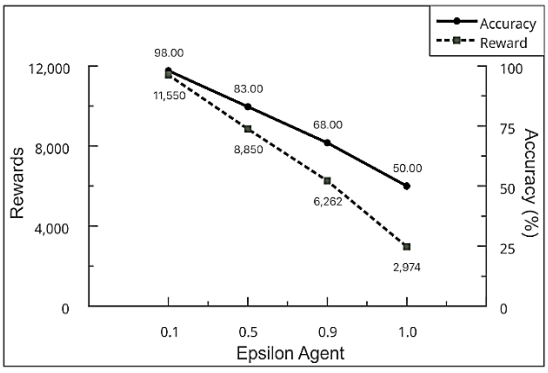

Figure 4. Accuracy and reward comparison between agents

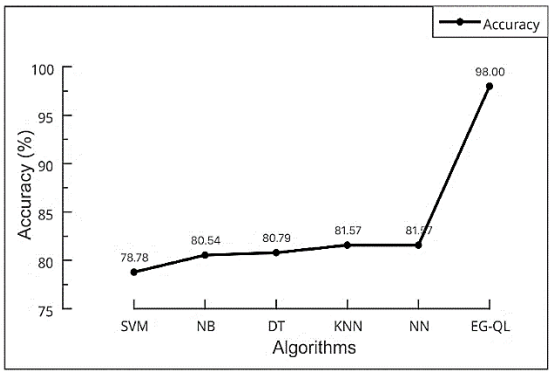

Figure 5. Accuracy comparison between algorithms

Based on the result of Figure 5, this study compared several algorithms like support vector machine (SVM), naive Bayes (NB), decision tree (DT), k-nearest neighbor (KNN), neural network (NN), and epsilon greedy optimized Q learning (EG-QL). Compared to other machine learning models, the proposed Q learning agent has the highest accuracy of 98%. This means that the proposed model has the best performance compared to other models. Followed by NN with 81.57%, KNN with 81.57%, DT with 80.79%, naïve Bayes with 80.54%, and SVM up to 78.78%

The last aspect of the evaluation is the performance benchmark for the Raspberry Pi device. In this part, this study split the performance benchmark into two parts: CPU and memory usage. The CPU usage result of the agents are illustrated in Figure 6.

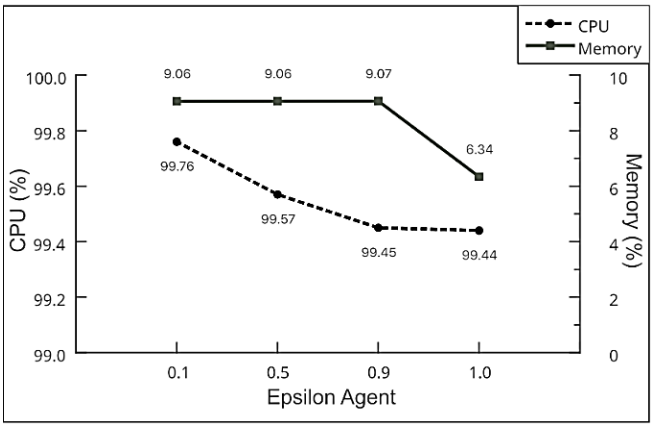

Figure 6. Performance benchmark on an SBC

According to Figure 6, all agents utilized more than 99% to process the data in the training and test phases. The process of the agent is in a single processor, so there are three more processors available for the operating system to use. If calculated roughly, the agent only used 25% of all processors available in the Raspberry Pi. Hence, the process itself will not disturb the whole system. According to the result, most agents have similar memory usage except agent 1.0. The dataset inside the agent caused the high memory usage in each agent. Besides that, the agent also stored the learning policy (Q Table) inside the agent. Thus, storing the learning policy also increased the memory usage in every agent (Agent 0.1, 0.5, and 0.9).

The last part of this section discusses the result of the agents' evaluation and comparison. The discussion covers the accuracy of the detection agents, the impact of the dataset on training and test processes, and the performance benchmark in the Raspberry Pi device. In the detection accuracy evaluation phase, this study compared Q learning agents with each other and the previously available models. The first comparison found that the best agent has the highest accuracy compared to other agents. In this case, the agent with epsilon configured to 0.1 has the best accuracy up to 98%. The agent can reach the top accuracy because the agent used the best policy more often than random action space. Using the best policy as the main source of action can give the agent more proper choice than depending on the randomization. Thus, the agent can reach maximum accuracy faster than other agents. Reward evaluation determines how well the agent detects the intrusion. Similar to the accuracy test, a higher reward is always preferable to others. In this case, the agent with epsilon 0.1 has the highest reward with 11,550. Followed by agent 0.5 with 8,850 rewards, agent 0.9 with 6,262 rewards, and agent 1.0 with 2,974 rewards. Agent 1.0 in the evaluation phase has the lowest rewards since the agent relies on randomness to detect the intrusions.

The second comparison was the top agent with other machine learning algorithms. According to the second comparison's result, the epsilon greedy optimized Q learning agent has the highest accuracy. Then, followed by NN, KNN, DT, naive Bayes, and SVM. There are several reasons why the agent has the highest accuracy compared to other models. One of the reasons is also related to the dataset used in the training and test phases. The dataset used to teach the agents consists of two components: neutral and intrusion data. No matter what type of attack is inside the dataset, the number is the most important factor. The balanced number can prevent the agent from siding on the heavier side after the training process. To do that, this study used stratified data split process to make sure the data is balanced. The next factor is the test data used in the testing process. Since the agent learned everything in the training process, the agent already has the best policy for each test data. However, if new unknown data is added to the test data the accuracy could decrease.

The last discussion is the performance benchmark on the Raspberry Pi device. The purpose of the evaluation is to test the agent's feasibility in an embedded device. According to the performance benchmark, the agents used more than 99% in a single processor to run the whole process. From a single processor point of view, this is a bad practice and not feasible to implement the agent in a real situation. If the agent is installed on a device with multiple processors, the system still has three more processors available. The last performance benchmark is memory usage. This aspect evaluated the memory usage of the agents during the whole process. The usage of each agent is affected by the dataset used. It means that the more dataset used in the agent, the more memory will be used. Agent 1.0 is an exception because the agent did not split the data into training and testing data. Thus, did not increase memory usage. In the terms of feasibility of memory usage, all the agents can run normally inside the Raspberry Pi without hindering the operating system. It can be concluded that the proposed algorithm and its agent can determine whether the packet data is an intrusion or not correctly. Compared to control models from a previously published article, the proposed agent has the best accuracy among the models. Besides that, the agent has lower system specifications and is feasible on the internet of things device.

5. CONCLUSION

Network security is a vital aspect of this modern era. Since many devices are connected to the internet, security protection is a serious concern. One technology that depends on network connectivity is IoT. The IoT device is connected to the internet and exposed to the invisible risk of attack. Besides that, the use of IPv6 as the communication protocol also posed an additional risk to the devices. To mitigate this problem, this study proposed an intrusion detection system using reinforcement learning. According to the evaluation results, the Q learning detection agent 0.1 outperformed the other agents' accuracy and rewards. With up to 98% of accuracy and 11,550 rewards, agent 0.1 has the highest accuracy compared to other agents. If compared to control models from the published article, the current agent is still in the first place. The current agent has an accuracy of up to 98%, followed by NN with 81.57%, KNN at 81.57%, DT at 80.79%, NB at 80.54%, and SVM up to 78.78%. Besides accuracy, the agent is also evaluated for the performance benchmark to test its feasibility. According to the performance benchmark, the agent has the highest CPU usage with more than 99% and memory usage up to 9.96%. However, in multi-processor devices, this is not a big problem. Hence, the agent is feasible to be installed on Raspberry Pi devices only.

ACKNOWLEDGEMENTS

The authors are grateful to the Education and Culture Ministry Republic of Indonesia for supporting this study through the Grant Research World Class Professor in 2021.

REFERENCES

[1] A. Shiranzaei and R. Z. Khan, "IPv6 security issues - a systematic review," in *Next-Generation Networks*, 2018, pp. 41–49, doi: 10.1007/978-981-10-6005-2_5.

[2] A. A. Bahashwan, M. Anbar, and S. M. Hanshi, "Overview of IPv6 based DDoS and DoS attacks detection mechanisms," in *ACeS 2019: Advances in Cyber Security*, 2020, pp. 153–167, doi: 10.1007/978-981-15-2693-0_11.

[3] L. Ubiedo, T. O'Hara, M. J. Erquiaga, and S. Garcia, "Current state of IPv6 security in IoT," *arXiv:2105.02710*, May 2021.

[4] I. Butun, P. Osterberg, and H. Song, "Security of the internet of things: Vulnerabilities, attacks, and countermeasures," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 616–644, 2020, doi: 10.1109/COMST.2019.2953364.

[5] M. A. M. Sadeeq, S. R. M. Zeebaree, R. Qashi, S. H. Ahmed, and K. Jacksi, "Internet of things security: a survey," in *2018 International Conference on Advanced Science and Engineering (ICOASE)*, Oct. 2018, pp. 162–166, doi: 10.1109/ICOASE.2018.8548785.

[6] O. Yousuf and R. N. Mir, "A survey on the internet of things security," *Information and Computer Security*, vol. 27, no. 2, pp. 292–323, Jun. 2019, doi: 10.1108/ICS-07-2018-0084.

[7] E. A. Shammar and A. T. Zahary, "The internet of things (IoT): a survey of techniques, operating systems, and trends," *Library Hi Tech*, vol. 38, no. 1, pp. 5–66, Oct. 2019, doi: 10.1108/LHT-12-2018-0200.

[8] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan, "Internet of things (IoT) security: current status, challenges and prospective measures," in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, Dec. 2015, pp. 336–341, doi: 10.1109/ICITST.2015.7412116.

[9] A. Triantafyllou, P. Sarigiannidis, and T. D. Lagkas, "Network protocols, schemes, and mechanisms for internet of things (IoT): Features, open challenges, and trends," *Wireless Communications and Mobile Computing*, vol. 2018, pp. 1–24, Sep. 2018, doi: 10.1155/2018/5349894.

[10] S. Zeadally and M. Tsikerdekis, "Securing internet of things (IoT) with machine learning," *International Journal of Communication Systems*, vol. 33, no. 1, Jan. 2020, doi: 10.1002/dac.4169.

[11] A. Sforzin, F. G. Marmol, M. Conti, and J.-M. Bohli, "RPiDS: Raspberry Pi IDS - a fruitful intrusion detection system for IoT," in *2016 IEEE Conferences on Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, Jul. 2016, pp. 440–448, doi: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-

SmartWorld.2016.0080.
[12] T. Sommestad, H. Holm, and D. Steinvall, "Variables influencing the effectiveness of signature-based network intrusion detection systems," *Information Security Journal: A Global Perspective*, vol. 31, no. 6, pp. 711–728, Nov. 2022, doi: 10.1080/19393555.2021.1975853.
[13] T. Jiang, J. L. Gradus, and A. J. Rosellini, "Supervised machine learning: a brief primer," *Behavior Therapy*, vol. 51, no. 5, pp. 675–687, Sep. 2020, doi: 10.1016/j.beth.2020.05.002.
[14] I. H. Sarker, "Machine learning: algorithms, real-world applications and research directions," *SN Computer Science*, vol. 2, no. 3, May 2021, doi: 10.1007/s42979-021-00592-x.
[15] A. M. Karimi, Q. Niyaz, W. Sun, A. Y. Javaid, and V. K. Devabhaktuni, "Distributed network traffic feature extraction for a real-time IDS," in *2016 IEEE International Conference on Electro Information Technology (EIT)*, May 2016, pp. 0522–0526. doi: 10.1109/EIT.2016.7535295.
[16] T. Ahmad and M. N. Aziz, "Data preprocessing and feature selection for machine learning intrusion detection systems," *ICIC Express Letters*, vol. 13, no. 2, pp. 93–101, 2019.
[17] H. Malhotra and P. Sharma, "Intrusion detection using machine learning and feature selection," *International Journal of Computer Network and Information Security*, vol. 11, no. 4, pp. 43–52, Apr. 2019, doi: 10.5815/ijcnis.2019.04.06.
[18] M. Anbar, R. Abdullah, B. N. Al-Tamimi, and A. Hussain, "A machine learning approach to detect router advertisement flooding attacks in next-generation IPv6 networks," *Cognitive Computation*, vol. 10, no. 2, pp. 201–214, Apr. 2018, doi: 10.1007/s12559-017-9519-8.
[19] M. Zolanvari, M. A. Teixeira, L. Gupta, K. M. Khan, and R. Jain, "Machine learning-based network vulnerability analysis of industrial internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6822–6834, Aug. 2019, doi: 10.1109/JIOT.2019.2912022.
[20] P. Winder, *Reinforcement learning: Industrial applications of intelligent agents*. O'Reilly Media, 2020.
[21] M. Schrötter, T. Scheffler, and B. Schnor, "Evaluation of intrusion detection systems in IPv6 networks," in *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications*, 2019, pp. 408–416, doi: 10.5220/0007840104080416.
[22] W. Dabney, G. Ostrovski, and A. Barreto, "Temporally-extended ε-greedy exploration," *arXiv:2006.01782*, Jun. 2020.
[23] M. Kubat, *An introduction to machine learning*. Cham: Springer International Publishing, 2017, doi: 10.1007/978-3-319-63913-0.
[24] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. 7, pp. 133653–133667, 2019, doi: 10.1109/ACCESS.2019.2941229.
[25] J. Clifton and E. Laber, "Q-learning: theory and applications," *Annual Review of Statistics and Its Application*, vol. 7, no. 1, pp. 279–301, Mar. 2020, doi: 10.1146/annurev-statistics-031219-041220.
[26] A. A. Salih and A. M. Abdulazeez, "Evaluation of classification algorithms for intrusion detection system: a review," *Journal of Soft Computing and Data Mining*, vol. 02, no. 01, Apr. 2021, doi: 10.30880/jscdm.2021.02.01.004.
[27] S. Manickam *et al.*, "Labelled dataset on distributed denial-of-service (DDoS) attacks based on internet control message protocol version 6 (ICMPv6)," *Wireless Communications and Mobile Computing*, vol. 2022, pp. 1–13, 2022, doi: 10.1155/2022/8060333.

BIOGRAPHIES OF AUTHORS

April Firman Daru is currently studying for a Doctor of Computer Science at the Universitas Kristen Satya Wacana (UKSW) Salatiga, Indonesia. Since 2010 until now he has been a Lecturer in the Informatics Engineering Study Program at the Universitas Semarang (USM). His research interests are in the fields of internet of things, artificial intelligence, attack detection and machine learning. He can be contacted at email: firman@usm.ac.id.

Kristoko Dwi Hartomo completed his doctorate in the Doctorate Program of Computer Science, Science Faculty of Gadjah Mada University in 2017. He has been active in research since 2008 until now on intrusion detection systems, spatial data processing and remote sensing. He has published his papers in international journals, has 5 copyrights, and wrote some reference books on remote sensing data analysis and modelling in Indonesia and English. He can be contacted at email: kristoko@uksw.edu.

Hindriyanto Dwi Purnomo is an Associate Professor at Department of Information Technology, Universitas Kristen Satya Wacana, Indonesia. He received his bachelor's degree in engineering physics, from Gadjah Mada University, Indonesia, in 2005, Magister of Information Technology from The University of Melbourne, Australia in 2009 and Doctor of Philosophy in Industrial and System Engineering from Chung Yuan Christian University, Taiwan, in 2013. He has received several recognitions and awards for his academic achievement. He also has published many articles in reputable journals, conferences, and books. His research interests are in the field of metaheuristics, soft computing, machine learning and deep learning. He can be contacted at email: hindriyanto.purnomo@uksw.edu.