Leveraging Reinforcement Learning for Adaptive Monitoring of Low-Power IoT Networks

Mohamed Said Frikha*, Abdelkader Lahmadi†, Sonia Mettali Gammar*, Laurent Andrey†
*CRISTAL LAB, National School of Computer Sciences, ENSI, Manouba, Tunisia,
medsaid.frikha@ensi-uma.tn, sonia.gammar@ensi.rnu.tn
† Université de Lorraine, CNRS, Inria, Loria, F-54000 Nancy, France,
{abdelkader.lahmadi|laurent.andrey}@loria.fr

***Abstract*—Low-power Internet of Things (IoT) networks are widely deployed in various environments with resource-constrained devices, making their states monitoring particularly challenging. In this paper, we propose an adaptive monitoring mechanism for low-power IoT devices, by using a reinforcement learning (RL) method to automatically adapt the polling frequencies of the collected attributes. Our goal is to minimize the number of monitoring packets while keeping accurate and timely detection of threshold crossings associated with supervised attributes. We study the various RL parameter settings under different monitoring attribute behaviors using the OpenAi Gym simulator. We implement the RL based adaptive polling in Contiki OS, and we evaluate its performance using the Cooja simulator. Our results show that our approach converges to optimal polling frequencies and outperforms static periodic notification-based methods by reducing the number of monitoring packets, with a percentage of correctly detected threshold crossings exceeding 80%.**

***Index Terms*—Low-power networks, Internet of Things (IoT), Reinforcement Learning (RL), adaptive monitoring.**

I. Introduction

Low-power Internet of Things (IoT) networks have been extremely deployed in various applications such as e-health, transportation systems, smart home, and environmental monitoring. They rely on resource-constrained devices with low processing, memory, and communication capabilities due to the limited battery capacity. Extending devices' lifetime is a fundamental requirement since they can be deployed in unreachable areas, which makes the replacement or the recharge of their batteries difficult or costly. However, energy is mostly consumed by the communication module as a result of the activation of antennas to transmit, receive, and listen to the radio channels. Thus, every packet transmitted in the network has a significant impact on the nodes' lifetime, in particular when the transmission of a packet is hop by hop.

To keep these networks operational and useful, multiple management tasks such as detecting anomalies, making decisions about resource allocation, job scheduling, and controlling the quality of service have to be carried by using an adapted network monitoring system. This system should be able to detect, precisely and with low overhead, threshold crossings associated with each monitored attribute. To meet this tradeoff, existing solutions rely on hand-tuned polling frequencies, which is a tedious task. Quite frequently, the monitoring of these devices consists of programming them to periodically send monitoring packets to the management server with the new states of the monitored attributes, regardless of the node state or the evolution of the attributes over time [1]–[3]. Indeed, there is a lack of a general monitoring framework that can automatically adapt the polling frequencies of the monitored attributes in order to reduce its overhead and to extend the network lifetime.

Network monitoring systems have been extensively studied, in particular, to develop efficient algorithms with lower overhead or to adapt the thresholds of the monitored attributes. Wuhib et al. [4] proposed an adaptive aggregation protocol of monitoring data to detect threshold crossings, where nodes dynamically adjust their neighbor interaction rates, which considerably reduce communication overhead. Mijumbi et al. [5] developed DARN, a real-time monitoring system using machine learning and neural networks. The goal of DARN is to classify the monitoring metrics into clusters and then automatically generates and adapts the lower and upper baselines of the changes in network operating conditions. Dilman et al. [6] developed two reactive monitoring algorithms *simple-value* and *simple-rate* to detect threshold overruns. These solutions combine global polling with asynchronous trap events to reduce communication overhead in IP networks. In our work, we focus on dynamically adapting polling frequencies of monitoring systems to extend low-power IoT networks' lifetime.

There are also several research works that have applied reinforcement learning (RL) [7] to adapt the operations of wireless sensor networks (WSN) and preserve the batteries of their nodes. A new RL approach for energy harvesting in WSNs was applied in [8]. The Q-learning agent in the system takes the percent of energy left in the battery as a state to adjust the operating frequency mode. The goal is to exploit the collected information to adapt the node energy management policy relative to the harvested energy. Oddi et al. [9] proposed a Q-routing algorithm for fixed and mobile networks. By considering the current residual battery of one-step nodes, the proposed approach balances the routing effort. Moreover, it decreases the probability of sending feedback over time to reduce the network overhead. To ensure efficient security routing while preserving energy resources, an efficient secure routing algorithm based on Q-Learning (ESRQ) was proposed in [10]. The approach combines packet loss, data content,

distance, and residual energy as metrics. After forwarding each packet, the agent updates their trust values according to the collected metrics with the Q-Learning method for routing.

While there is a great amount of effort on applying RL technique on optimizing WSN operations to preserve energy, to the best of our knowledge, this technique has not been used to adapt and make decisions about the intervals of supervised attributes in network monitoring systems. In this paper, we formulate the network monitoring task as a learning process of the optimal polling periods for each monitoring attribute to detect its threshold crossings, without assuming any specific pattern on their behaviors. To validate our RL-based monitoring approach, we analyze its parameters through extensive simulations using OpenAI Gym simulator. We implement our approach within the IoT simulator Cooja running emulated devices, and we evaluate its performance in terms of the percentage of detected threshold crossings and the overhead presented as the average number of monitoring packets.

The rest of the paper is organized as follows. Section II describes the RL model and gives the implementation details. The simulation environment and the evaluation results of our RL approach are presented in section III. Conclusions are presented in section IV with some future research directions.

II. Monitoring System Design

In this work, we propose a monitoring system for low-power IoT networks. This system is based on a poller-pollee architecture [11], where each poller node sends requests to its pollee to collect values of the monitored attributes, as shown in Fig.1.

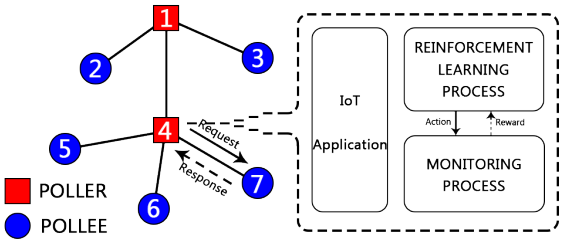


Fig. 1: A poller-pollee monitoring system where each poller employs an RL agent for adapting its polling frequency.

The poller node uses a monitoring process and an RL agent to adapt the polling intervals of the monitored attributes while considering their behaviors over time. The goal is to reduce the number of monitoring packets exchanged between a poller and its pollees by avoiding oversampling, and extend the network lifetime while providing the best possible percentages of detecting threshold crossings of the monitored attributes. To reach this goal, the RL agent must find the best polling intervals, without missing the time instants when the monitored attribute reaches or exceeds an upper threshold.

A. Reinforcement learning model

First, we detail the parameters and the components of the reinforcement learning model used by a poller to adapt the polling intervals.

1) States space (S): A state $s_i^t \in S$ of a node i at time t is the tuple of the current polling interval (SI), in seconds, selected by the agent, and the quality (Θ) of a monitored variable value during the last interval:

$$s_i^t = (SI_i^t, \Theta_i^t) \tag{1}$$

The polling interval SI_i^t takes discrete integer values in the range $[T_{min}, T_{max}] \subseteq \mathbb{N}^+$, while T_{min} is the sensor data sampling interval used by the deployed IoT application to send periodically a data packet to a sink node. Θ_i^t is a boolean variable that indicates the quality of the polled monitoring value specified by the monitoring requirements.

2) Actions space (A): An action is taken in each time step, which corresponds to setting the next polling interval for collecting the value of a monitored attribute. The agent can take one of three actions:

$$A = \{a_I, a_K, a_D\} \tag{2}$$

where a_I is the action to increase the polling interval, a_K is the action to keep the current interval and a_D is the action to decrease the polling interval.

To reduce the required memory space for the RL algorithm and especially avoid abrupt changes, actions can only move to the neighboring values of the polling interval. In the two particular states where $SI = T_{min}$ and action selected is a_D or $SI = T_{max}$ and action selected is a_I, the agent can not execute these actions. In such situations, the system keeps the same polling period.

The policy used by the system follows an ϵ-greedy strategy, such that the node selects the optimal action with a probability $1 - \epsilon$, and with a small exploration probability ϵ it chooses a random action.

3) Reward function: After the execution of each action, the agent obtains a new state of the environment as well as an immediate reward value, which evaluates the action taken in the previous state. In our monitoring system, the immediate reward represents the tradeoff between the cost and the quality of the polling interval. The reward function defined in our approach is:

$$r(s_i^t, a^t) = \frac{SI_i^t}{T_{min}} \times \beta_r \tag{3}$$

where $r(s_i^t, a^t)$ is the immediate reward returned in state s for a node i and taking action a at time t; β_r is a numerical representation of the monitored attribute quality during the polling interval.

The value of β_r takes 1 if no anomaly has been detected. The positive reward in this function represents the gain in packets rate sent by the monitored node, e.g. if the polling interval is $(T_{min} \times 8)$ seconds, the sensor node is transmitting eight times less than if it was in static polling with T_{min} as period. However, if the monitored attribute value reaches or exceeds the fixed threshold, a punishment $(-p)$ is returned as a value of β_r, such that p represents the number of threshold

crossings. Therefore, the reward value decreases as p increases to force the agent to react fast and reduce the polling interval.

4) Q-Value function: The value function is a numerical representation of how it's good of being in a future state for an agent. The Q-value is equal to the total rewards an agent can expect to accumulate starting from actual state and following a policy π. Many different methods are proposed in the literature to estimate the value of states. In this work, we are interested in three main Temporal-Difference (TD) [12] learning methods: SARSA [13], Q-learning [14], and Expected SARSA [15]. TD methods are based on a set of estimates to update their experiences without the need for a model of environmental dynamics to improve their learning. The Q-value function of the three TD methods, SARSA, Q-learning, and Expected-SARSA are defined as follows, respectively:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (4)$$

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (5)$$

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \sum_a \pi(a|s_{t+1}) Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (6)$$

where $Q(s_t, a_t)$ represents the estimate Q-value of taking action a in state s at time t; r_{t+1} represents the immediate reward returned at time $t+1$; $\max_a Q(s_{t+1}, a_{t+1})$ represents the estimate Q-value of taking the optimal action a in state s at time $t+1$; $\pi(a|s_{t+1})$ represents the probability for executing action a at state s under stochastic policy π; $0 < \alpha \leqslant 1$ is the learning rate; and $0 \leqslant \gamma \leqslant 1$ is the discount factor.

SARSA and its variant Expected-SARSA are an on-policy TD control methods. They use a stochastic policy to update their Q-values. This makes an important variance in the update function which can slow their convergence. The main difference between them is that Expected-SARSA computes an expectation taking into consideration each action under the current policy, represented by the part $\sum_a \pi(a|s_{t+1}) Q(s_{t+1}, a)$ in (6). Q-learning is an off-policy TD control method, where during the learning, the agent selects the optimal action, represented by the part $\max_a Q(s_{t+1}, a_{t+1})$ in (5), independently of the policy being followed.

B. Monitoring algorithm

The algorithm 1 describes the polling function which takes as input the action selected by the RL agent. It steps the environment one step ahead by computing the interval of the next polling action of the attribute value from the monitored node (line 1). During this interval, the monitored node checks periodically, with a period T_{min}, it's own monitored attribute values to detect all its threshold crossings. This number of local threshold crossings is used as feedback for the reward function. At the end of the interval, the agent sends a request message (line 3) to the monitored node to get the current attribute value. The pollee node sends a response message (line 4) with two fields: the last monitored attribute value and the number of times this attribute crossed its thresholds, which represents the polling quality used by the reward function. From this quality value, the agent obtains the new environment's state and the reward signal according to the action chosen in the previous state (lines 5 and 6).

Algorithm 1 RL-based polling function.

Require:
a: selected action by the RL agent
1: $t_s \leftarrow apply(a)$
2: $sleep(t_s)$
3: $send\_request\_packet(monitored\_node\_id)$
4: $attr\_quality, last\_attr\_val \leftarrow get\_response\_packet()$
5: $s_{t+1} \leftarrow get\_new\_state(attr\_quality)$
6: $r_{t+1} \leftarrow calculate\_reward\_value(s_{t+1})$
7: $info \leftarrow \{\}$
8: **if** $last\_attr\_val \geq threshold$ **then**
9: $\quad info \leftarrow \{"Alert: \ Threshold \ crossed!"\}$
10: **end if**
11: **return** $s_{t+1}, r_{t+1}, info$

III. Evaluation

In this section, we describe the simulation environment and its parameters used for simulating our RL based monitoring system as well as the details of the evaluation scenarios. The common parameters used both in simulation and experimental evaluations are shown in Table I.

TABLE I: Common simulation and experimental parameters

Parameter	Value
Experiment duration	$8000s$
T_{min}	$1s$
Polling Interval (SI) range	[T_{min}, $T_{min} \times 2$, $T_{min} \times 4$, $T_{min} \times 8$]
Monitored attribute threshold	60%

A. Simulation setup

To achieve the best performance of the approach, we must identify at first the optimal values for the different parameters that affect the operations of the RL algorithm. Four parameters have an impact on the learning process of the RL agent: the probability of exploration (ϵ), the learning rate (α), the discount factor (γ), and the Q-value function. We use OpenAi Gym [16] simulator (v0.13.1), which is a toolkit for developing and comparing RL algorithms, to evaluate the effect of these parameters on the performance of the RL agent under three different types of monitored attributes behaviors. The main parameters used in the simulations are shown in Table II.

The terms anomaly and threshold crossing are used interchangeably in the rest of the paper. Each RL configuration was simulated 3 times using a laptop with 2.50GHz i5 CPU and 4GB RAM running Ubuntu 18.04.

TABLE II: OpenAi Gym simulation parameters

Parameter	Value
Percentage of anomalies	5%
Exploration probability (ϵ) range	$[0.1 \dots 0.9]$
Learning rate (α) range	$[0.1 \dots 0.9]$
Discount factor (γ) range	$[0.1 \dots 0.9]$

1) Monitored attribute changes: A set of attribute values were generated to evaluate the RL agent performance under different attribute behaviors in the same controlled situations. The agent was tested on three different types of monitored attribute changes as proposed in [6]:

- **Uniform Change (Attr_UC)**: The variation of the monitored attribute Δy_i at time t, is a uniformly distributed random variable in a given range $[-k..k]$. In our case, k was chosen to be 2% of the range of the variable y_i. Fig.2a shows the variation of monitored variable values with a few randomly injected peaks.
- **Normal Change (Attr_NC)**: The variation of the monitored attribute Δy_i at time t, is a normally distributed random variable with a mean value of 0 and standard deviation k. k was fixed to be 1 for this simulation. Fig.2b illustrates the behavior of y_i in time.
- **Self-Similar (Attr_SS)**: The monitored variable y_i is a superposition of independent on/off sources that have ON-OFF times heavy-tailed. Fig.2c represents the shape of the variable y_i under this behavior.

We consider in this work, as an illustrative example, a monitoring variable that represents the CPU utilization of a pollee node. In the state space, the quality Θ is defined as good when the monitored CPU utilization is under a fixed threshold, which we set to 60% as suggested in [5]. If the monitored attribute value crosses this threshold, an anomaly is considered and has to be detected by the poller node.

The values of each attribute are generated for a duration of 8000 seconds. We assume that the states in the first 500 seconds are always normal with the attribute values under the fixed threshold. The percentage of anomalies (i.e., threshold crossings) in every attribute behavior was set to 5% of the duration.

2) Performance metrics: We evaluate the performance of the proposed monitoring approach, in terms of its accuracy (7), the percentage of detected anomalies (8), and the number of transmitted monitoring packets per node. The accuracy measures the ability of the RL agent to correctly decide to raise or not an alert in each time step. The percentage of detected anomalies, also known as recall in machine learning classification problems, measures its ability to correctly raise an alert for all occurred anomalies. These two metrics are defined as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{7}$$

$$Detected\_Anomalies = \frac{TP}{TP + FN} \tag{8}$$

where TP represents the True Positive (i.e., an alert is raised by the RL agent when an anomaly occurs); TN represents the True Negative (i.e., no alert is raised by the RL agent when no anomaly occurs); FP represents the False Positive (i.e., an alert is raised by the RL agent when no anomaly occurs); and FN represents the False Negative (i.e., no alert is raised by the RL agent when an anomaly occurs).

B. Simulation results

Our first evaluation is aimed at finding the best probability of exploration-exploitation (ϵ) of the RL agent for the different experiments and in each approach. For that, we calculate the percentage of the best experiments, in terms of accuracy, for each value of epsilon. We conducted a series of simulations, but due to lack of space, we didn't present the obtained figures. Our results show clearly that $\epsilon = 0.1$ gives the best percentage of accuracy in most experiments. Therefore, the RL agent does not require much exploration to find the best actions and to improve its performance to detect more threshold crossings. In the rest of this paper, we set ϵ to 0.1.

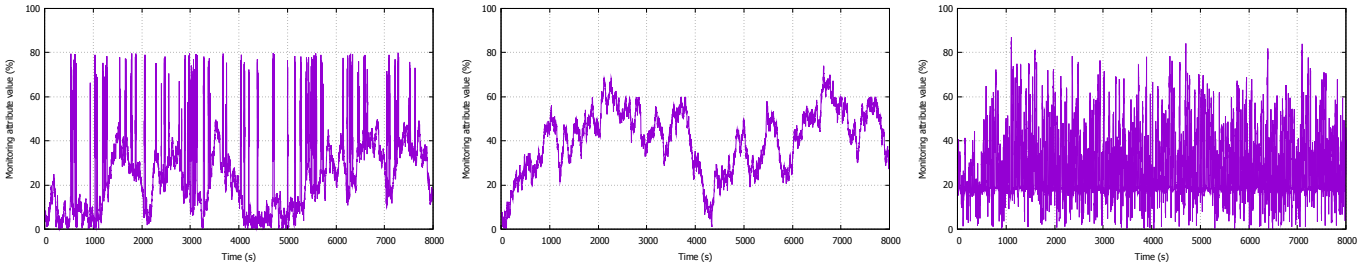
1) The effect of learning rate and discount factor: The values of the learning rate α and the discount factor γ affect the learning experience of the RL agent since they are used in the Q-value function. So in this part, we analyze the impact of the different values of the tuple (α, γ) on the accuracy and the average polling interval obtained at the end of each experiment with the different attribute behaviors. We notice that the accuracy is consistently greater than 95% because of a percentage of false positive always equal to 0%, which is one of the benefits of our proposed approach.

For the attributes Attr_UC and Attr_SS, the best results in terms of detected anomalies are obtained with the lowest learning rate (i.e., $\alpha = 0.1$) and higher discount factor values (i.e., $\gamma \in \{0.8, 0.9\}$). On the contrary, the outcomes with the normal change attribute (Attr_NC) show that the percentage of accuracy increases slightly with the learning rate and reaches the best rates when the values of γ are low.

Globally, the couple (α, γ) that makes the best result regarding the accuracy and the detected anomalies have the lowest average polling interval compared to the other configurations.

2) Convergence analysis: After evaluating the effect of the learning rate and discount factor, we study the performance of the TD methods in terms of the convergence rate of detected anomalies and polling intervals during the simulation period. For every attribute behavior, we consider the best couples (α, γ) that provide either the highest accuracy or average polling interval. In addition to the comparison of the three TD functions between them, we compare our approach with the following two network monitoring approaches:

- **Notification based monitoring**: In this approach, the pollees are programmed to send monitoring packets to the poller node at a fixed interval. The fixed interval values are in the range of the variable SI as specified in Table I. We denote these monitoring approaches as *static_1*, *static_2*, *static_4*, and *static_8* with the values 1, 2, 4 and 8 are their respective fixed interval values in seconds.



(a) Uniform change attribute. (b) Normal change attribute. (c) Self-similar change attribute.

Fig. 2: Monitoring attribute behaviors used in the simulation.

- **Optimal dynamic polling**: It represents the optimal polling intervals that the agent can select to obtain 100% accuracy and detected anomalies. In this approach, the agent has the capability to predict future attribute behavior and therefore it reduces the polling interval in advance to detect all anomalies. We denote this approach as *optimal_app*.

We observe from Table III that our RL based polling approach shows better detection performance compared to static notification based monitoring approaches (*static_2*, *static_4*, and *static_8*) for all attribute behaviors. However, when using *static_1*, we perceive an accuracy equal to 100% because the monitoring attribute values are collected with the lowest interval T_{min} which is 1 second.

TABLE III: Best percentage of detected anomalies with different attribute behaviors and monitoring configurations.

	static_2	static_4	static_8	RL approach
Attr_UC	48.63%	25.00%	12.22%	86.53%•
Attr_NC	49.64%	25.30%	12.90%	83.21%⋆
Attr_SS	51.21%	24.64%	12.32%	91.55%•

⋆ Best percentage is obtained by Q-learning
• Best percentage is obtained by Expected-SARSA

Since the best accuracy performances in Attr_UC and Attr_SS are obtained with low α and high γ values, the average polling interval converges linearly to the best solutions which increase the number of detected anomalies. With these two attribute behaviors, the average polling interval is below 2 seconds, which is also the monitoring interval of the static_2 application. The agent in these cases converges too far from the 5 seconds average polling interval achieved by the optimal_app. This result can be explained by the fact that the agent is interested more by its accumulated experience and future estimation more than the immediate reward, and so it does not react quickly to states' change. With Attr_NC behavior, the average polling interval converges nearly to the results of the optimal_app. The best accuracy performance is obtained with a high α and low γ values, so for this attribute behavior, the agent focuses on the immediate reward which represents the current environment state. Fig.3 illustrates how the agent adapts the polling interval according to the Attr_NC behavior when it reaches the convergence. After just a few seconds of the first threshold crossing by the attribute, the RL agent decreases the polling interval to collect more monitoring values. Even the attribute value decreases below the threshold, the RL agent keeps the same polling interval for a while, to be sure that the state is still good, then it gradually increases the interval.

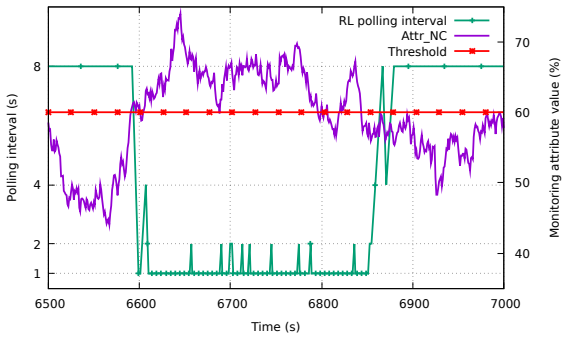


Fig. 3: Adaptation of the polling interval by the RL agent according to the Attr_NC attribute behavior using Q-learning TD function.

We found that the Q-learning method converges better, compared to SARSA and Expected-SARSA, towards both the optimal accuracy percentage and the polling interval values obtained by the *optimal_app*.

C. Implementation and experimental results

We implemented a prototype of our RL-based monitoring system with Contiki OS (v3.0) and we evaluate its performance using emulated nodes in the simulation environment Cooja [17]. We used the sensor platform Zolertia z1 [18] for motes type and the protocol IEEE 802.15.4 for the physical layer. In every monitored node (pollee), we implement a *stress process* to randomly increase the CPU utilization, which is our monitored attribute, during the experiment at different time instants and with different behaviors. Therefore, each pollee node has a different CPU behavior (i.e., anomalies distribution over time) and a random number of threshold crossings. Note that in these experiments, we are interested to detect anomalies with long-duration threshold crossings, as the case of Attr_NC behavior, rather than short duration threshold crossings using

the following settings: $\epsilon = 0.1$, $\alpha = 0.9$, $\gamma = 0.1$, and the Q-learning TD method. The evaluation results are the average of 10 RL agents recorded with various monitored pollee.

We observe that the percentage of detected anomalies of all agents is greater than 60%. The RL agents improve their knowledge to detect long-duration anomalies from the first seconds. If the threshold crossings are spanned for a long period, the agent reduces gradually the polling interval and detects more anomalies. Whereas if the number of anomalies is very low or distributed in time, the percentage of detected anomalies is always less than 20%.

We also noticed that in real networks, the RL agent can miss the detection of some anomalies, even it's using a T_{min} polling period. This problem is due to the time required to transmit the request packet, the processing time in the monitoring node to prepare the response data, the time until the agent receives them, and selects the next action. The sum of these delays can exceed T_{min} in some cases. Therefore, the monitoring node checks its own monitored attribute values more than the RL agent expects, which leads to a reduction in the percentage of detected anomalies.

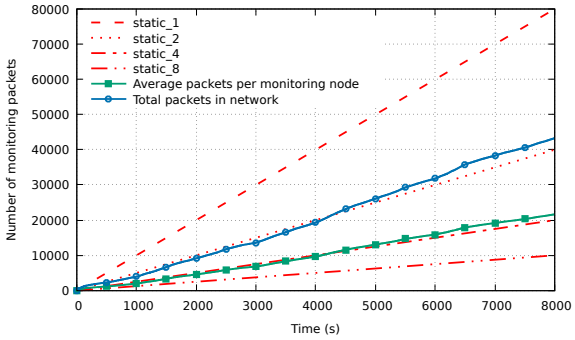


Fig. 4: Number of monitoring packets exchanged overtime under different evaluation scenarios.

As shown in Fig.4, the number of packets sent per the monitoring nodes of the RL-based monitoring approach is lower than *static_1* and *static_2* by approximately 73% and 55%, respectively. In terms of network bandwidth consumption, our RL approach uses a polling process (request/response) to collect monitoring data, compared to notification based applications which are transmitting monitoring values by a single message. Thus, the number of packets in the network for our approach is theoretically twice the packets used by notification based monitoring. However, the RL agents in our experiments detected almost 11% more anomalies than *static_2* with only 8% more packets in the network.

IV. Conclusion and Future Work

In this paper, we formulated the monitoring of low-power IoT networks as a reinforcement learning (RL) problem to dynamically adapt the polling interval of supervised attributes. We implemented and evaluated our RL based monitoring approach using OpenAi Gym RL and Contiki's Cooja simulators to measure its accuracy for detecting threshold crossings that represent anomalies, and its cost in terms of the number of monitoring packets. Our results show that the performance of the proposed approach varies mainly according to the attribute behavior, in particular how the anomalies are distributed over time, and the settings of the learning rate and discount factor. Furthermore, our approach outperforms fixed interval-based monitoring systems in terms of the percentage of detected threshold crossings and the number of packets exchanged in the network which conserves nodes' energy.

In future work, we will extend our approach with a multi-agent reinforcement learning technique, where each agent exchanges its local monitoring experience with other nodes to achieve global optimization and improve the monitoring accuracy and cost.

References

[1] R. Zhang, D. Yuan, and Y. Wang, "A health monitoring system for wireless sensor networks," in *2nd IEEE Conference on Industrial Electronics and Applications*. IEEE, 2007.

[2] J. Maerien, P. Agten, C. Huygens, and W. Joosen, "FAMoS: A flexible active monitoring service for wireless sensor networks," in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2012.

[3] A. Meier, M. Motani, H. Siquan, and S. Künzli, "DiMo: Distributed node monitoring in wireless sensor networks," in *11th international symposium on Modeling, analysis and simulation of wireless and mobile systems*. ACM, 2008.

[4] F. Wuhib, R. Stadler, and M. Dam, "Gossiping for threshold detection," in *IFIP/IEEE International Symposium on Integrated Network Management*, June 2009.

[5] R. Mijumbi, A. Asthana, M. Koivunen, F. Haiyong, and Z. Norman, "DARN: Dynamic baselines for real-time network monitoring," in *4th IEEE Conference on Network Softwarization*, 2018.

[6] M. Dilman and D. Raz, "Efficient reactive monitoring," *IEEE journal on selected areas in communications*, vol. 20, no. 4, pp. 668–676, 2002.

[7] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2017, http://incompleteideas.net/book/RLbook2018.pdf.

[8] Y. Rioual, J. Laurent, E. Senn, and J.-P. Diguet, "Reinforcement learning strategies for energy management in low power iot," in *International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2017.

[9] G. Oddi, A. Pietrabissa, and F. Liberati, "Energy balancing in multi-hop wireless sensor networks: an approach based on reinforcement learning," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, 2014.

[10] G. Liu, X. Wang, X. Li, J. Hao, and Z. Feng, "ESRQ: An efficient secure routing method in wireless sensor networks based on Q-learning," in *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications (TrustCom/BigDataSE)*. IEEE, 2018.

[11] A. Lahmadi, A. Boeglin, and O. Festor, "Efficient distributed monitoring in 6lowpan networks," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM)*. IEEE, 2013.

[12] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[13] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, England, 1994, vol. 37.

[14] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, 1992.

[15] H. Van Seijen, H. Van Hasselt, S. Whiteson, and M. Wiering, "A theoretical and empirical analysis of expected sarsa," in *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2009.

[16] "Gym: A toolkit for developing and comparing reinforcement learning algorithms," 2019. [Online]. Available: https://gym.openai.com/

[17] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *IEEE Workshop on Practical Issues in Building Sensor Network Applications*, 2006.

[18] "Zolertia z1 motes - contiki wiki," 2019. [Online]. Available: https://github.com/contiki-os/contiki/wiki/Zolertia-z1-motes