Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Predictive mobility and cost-aware flow placement in SDN-based IoT networks: a Q-learning approach

Gan Huang[1†], Ihsan Ullah[2†], Hanyao Huang[3] and Kyung Tae Kim[4*]

Abstract

Software-Defined Networking (SDN) has emerged as an innovative networking method that offers effective management and remarkable flexibility. However, current SDN-based solutions primarily focus on static networks or concentrate on backbone networks, where network dynamics have minimal impact. The existing methods for placing flow entries in Software-Defined Networking (SDN)-based Internet of Things (IoT) systems exhibit shortcomings in accurately predicting outcomes and efficiently reducing table misses and associated performance metrics. This research introduces a new approach, specifically the mobility-aware adaptive flow entry placement scheme for SDN-based Internet of Things (IoT) environments, to address the mobility aspect of networks. The proposed scheme utilizes the Q-learning algorithm to predict the next possible location of end devices, while the cost-sensitive AdaBoost algorithm is employed to select heavy and active flows. As a result, efficient flow rules for incoming flows can be dynamically implemented without controller intervention. Extensive computer simulations demonstrate that this approach significantly enhances match probability and prediction accuracy while concurrently reducing the number of table misses and resource expenditure compared to existing schemes.

Keywords SDN, OpenFlow, Flow rule placement, Q-learning, Mobility, Matching probability

Introduction

The development of Software-Defined Networking (SDN) aims to enhance the controllability of communication networks by decoupling the data plane from the control plane [1, 2]. In this context, SDN centralizes network management, facilitates programmability, and accelerates innovation through SDN abstraction. As illustrated in Fig. 1, the SDN structure incorporates the use of OpenFlow as a communication interface between the control plane and data plane, with the OpenFlow switch utilizing flow tables to process incoming packets [3–5]. In the control plane, the OpenFlow controller plays a crucial role in managing the entire network with a global view and defining how to handle flows using the flow table. Users' requirements are abstracted at the application layer, which interacts with the controller via the northbound interface. The OpenFlow switch performs a lookup using one or more flow tables and forwards the flow to the controller through the channels [6–9].

Nowadays, the utilization of the SDN framework is extensively implemented in the field of the Internet of Things (IoT). SDN empowers IoT by enabling dynamic management of networks, optimization of resources, and prioritization of traffic, thus accommodating the diverse

†Gan Huang and Ihsan Ullah contributed equally to this work.

*Correspondence:
Kyung Tae Kim
kyungtaekim76@gmail.com
[1] School of Mathematics and Computer Science, Zhejiang A&F University, Hangzhou 311300, China
[2] Advanced Technology Research Center, Korea University of Technology and Education, Cheonan, Republic of Korea
[3] Department of Computer and Software Engineering, Nanyang Institute of Technology, Nanyang 473004, China
[4] College of Computing and Informatics, SungKyunKwan University, Suwon, Republic of Korea

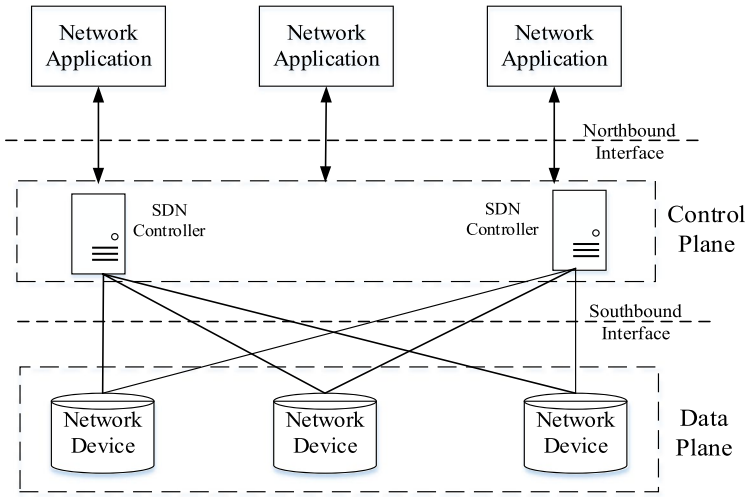


Fig. 1 The structure of SDN

and constantly evolving demands of IoT devices. The centralized control, security attributes, and ability to reduce latency make SDN an essential enabler for efficient and reliable implementations of IoT [10]. However, the broad implementation of the Internet of Things (IoT) presently generates enormous volumes of data from numerous devices linked to SDN [11]. The management of such extensive data becomes a significant challenge, especially when data types and characteristics are diverse [12]. Furthermore, it is essential to facilitate real-time transactions for time-critical applications. Delays in communication for such applications may result in catastrophic outcomes or significantly impede performance. Several approaches have been proposed to address the real-time challenge of SDN [13]. In SDN, incoming packets are managed based on the flow rule maintained in the switch's flow table. The controller can reactively (in response to arriving packets) or proactively add, update, and delete table rules. The controller can probe and handle flow entries at any time. Incoming packets matched with a flow entry are processed by the instructions kept in the entry. The probability of a table hit is dependent on the table configuration. If a flow entry handling table-miss exists, the instructions in that entry determine how to handle unmatched packets [14, 15]. The instruction options include dropping, transferring to another table, or communicating with the controller over the control channel via the packet-in message, resulting in a new flow entry. By default, flow entries for table-miss do not exist [16, 17]. A switch configuration can, however, override the default flow table and specify alternative behavior using the OpenFlow configuration protocol. As IoT devices become more widespread, centralized controllers will become increasingly crucial within the realm of Software-Defined Networking (SDN), particularly in terms of enhancing scalability and ensuring reliability [18].

The utilization of OpenFlow has facilitated the enhancement of network management to become more adaptable and efficient. However, the present implementation of Software-Defined Networking (SDN) still encounters limitations in dealing with diverse data and the continuously changing connectivity of Internet of Things (IoT) applications, particularly in a mobile setting [19, 20]. Furthermore, the current SDN-based flow rule placement schemes make assumptions either about the static nature of the network or the backbone networks, which in turn leads to a decrease in the dynamism of end devices. In the typical IoT environment, both stationary and mobile devices are deployed in everyday life [21]. Hence, the flow table must be dynamically updated in such an environment by enabling the switch to exchange user information with the backbone network. Given this context, it is crucial to devise a novel model for flow rule placement that takes into account user mobility [22]. While two viable approaches for flow rule placement exist, namely reactive and proactive, achieving the optimal balance between them represents a central focus of research.

This paper proposes a novel mobility-aware flow rule placement scheme that minimizes communication delay by considering the mobility of the nodes. It capitalizes on the merit of both the reactive and proactive flow rule placement approach. In the proposed scheme, the position of the end devices is predicted using the Q-learning (QL) algorithm to alleviate the computation overhead of the switch [23]. The controller decides the placement of flow entries based on the prediction. The AdaBoost scheme is employed to classify the flow, which also helps

the Q-learning in predicting the position of the nodes [24]. NS3 and Matlab evaluate the efficiency of the proposed scheme. It reveals that the proposed scheme significantly increases the number of high-level flow rules in the flow table compared to the existing schemes. Furthermore, it indicates that the communication delay and load of the controller are substantially decreased. The main contributions of the proposed scheme, called mobility-aware flow rule placement with Q-learning (MAPQ), are as follows.

- Even though AdaBoost has long been used in various fields, little research has been reported on efficiently and adequately deciding a strong classifier. A novel approach is developed which effectively achieves the final classifier considering the cost of error of the classification. Here the flows are classified into two types.
- The QL algorithm is combined with the cost-sensitive AdaBoost in the proposed MAPQ scheme, and the approach is generic such that it can be readily applied to other application problems employing AdaBoost to improve performance.
- A scheme is developed that predicts the position of the end node using the QL algorithm to select the flow entry that needs to be appropriately adjusted. Here the flows are classified based on the usage history to reduce the computation overhead required for proactive prediction.
- A new approach based on the QL algorithm is developed in which the next state and the corresponding reward are not obtained as soon as a specific action is chosen but after some event occurs. This approach reduces computation overhead while supporting high accuracy of prediction with QL.

The rest of the paper is structured as follows. Section 2 discusses the work related to flow rule placement for SDN, and Section 3 presents the proposed scheme based on cost-sensitive AdaBoost and QL. In Section 4, the performance of the proposed scheme is presented through simulation results and subsequently contrasted with the current approaches. Finally, Section 5 provides a conclusive summary of the paper with some remarks.

Related work

The basic terminologies of SDN are defined as follows:

***Definition* 1** (*Flow entry*). This flow entry constitutes the fundamental element of a flow table, encompassing the domain that specifies the circumstances required for a match (Match Field), direction pertaining to actions for the incoming packet that has been matched (Instructions), a record of statistics associated with matched packets (Counter), the maximum duration or idle time prior to the expiration of an entry (Time-out), among others. Table 1 showcases the basic fields of a flow entry.

***Definition* 2** (*Hard time-out,* T_H). Once a flow entry has been assigned to the flow table, it is subject to eviction upon the expiry of T_H, regardless of the number of matches.

***Definition* 3** (*Idle time-out,* T_I). A flow entry must be removed from the flow table when the Time Interval TI has elapsed since the last match.

***Definition* 4** (*Stale entry*). The entry is unmatched before T_I expires.

***Definition* 5** (*Elephant flow*). This large flow takes up more than a certain percentage of the link traffic during a given time interval [25]. Assume a certain length of time, *t*. Large flows are those sending packets more than a given threshold (say *P*%) in the previous period of *t*, where *P* is usually 0.1% of the link capacity and its corresponding flow entry is a long-lived entry.

***Definition* 6** (*Mice flow*). Contrary to elephant flow, mice flow stands for a flow taking less than *P*% of the link capacity in the previous period of *t*. In SDN, the majority of flows are mice flow.

A comprehensive investigation of multiple extant works concerning the placement of flow rules for SDN has been explicated in reference [19, 20]. The current schemes for flow rule placement primarily address two main obstacles: resource constraint and signaling overhead. Concerning the matter of resource constraint, two methods have been suggested, namely, flow eviction and flow compression.

Flow eviction

The researchers were motivated to develop effective flow eviction methods due to the flow table's limited size and declining performance caused by increasing stale entries. Openflow network implements a time-out mechanism to flush flow entries, but different time-out values have been used, ranging from 5 [26] to 60 seconds [22]. However, a fixed time-out value is inadequate for dynamically changing flows. To address this issue, the Adaptive Hard Time-out Method (AHTM) [27] was proposed, which utilized the M/G/C/C/FCFS queuing system to model the flow table and analyze truncation time and blocking probability to determine the optimal value of T_H.

Table 1 The basic fields of a flow entry

Match Field	Priority	Counter	Instruction	Time-out	Cookie

In the publication by Zhang et al. [28], another scheme targeting dynamic T_H was proposed in which the flow is classified into four kinds and the flow duration time, flow type, and utilization of the flow table were taken into account to determine its value. SmartTime [29] combined the adaptive idle time-out scheme with a proactive eviction strategy and utilized the Ternary Content Addressable Memory (TCAM) to minimize the number of table misses. Babangida et al. [30] proposed an algorithm known as idle-hard timeout allocation (IHTA) with the aim of enhancing flow table management in software-defined networking (SDN). The IHTA algorithm adaptively modified idle and hard timeout values for various flows by considering traffic patterns and packet inter-arrival times.

DIFANE [31] proposed a distributed flow management architecture that efficiently handles wildcard rules and quickly reacts to network dynamics such as policy/topology change and host mobility. Instead of the time-out approach, Multiple Bloom Filters (MBF) [32] were used for logging data, where the importance of each flow entry was encoded and the entry with the lowest importance value was evicted upon a table miss. Flowmaster [33] used a learning predictor based on the Markov model to identify flow entries expected to become idle and evict an existing flow entry based on the number of transitions and transition probability using the time-out mechanism. A proactive approach was proposed based on predicting the probability of matching entries using HMM [34]. The scheme in the publication by Huang et al. [35] employed HMM and fuzzy logic to select the victim entry proactively.

Flow compression

Applying a wildcard mechanism in compression reduces the number of flow entries present in a flow table while preserving the original semantics. This process involves the aggregation of specific flow entries into a single entry if they share common characteristics. Notably, the OpenFlow protocol does not require adjustment during flow compression.

Optimal Routing Table Constructor (ORTC) [36], which employed a binary tree to determine the minimal number of prefixes for each rule, represented one of the earliest methods for table aggregation. Once constructed, the routing table derived by ORTC remains unaltered. ORTC has also been implemented using structures other than binary trees [37].

The Palette distribution framework, introduced in the publication by Kanizo et al. [38], decomposed and aggregated large SDN tables into a predetermined number of smaller tables of the same semantics, which are then distributed across switches. In the publication by Iyer et al. [39], SwitchReduce, as a system, was proposed to optimize OpenFlow networks by reducing switch state size and controller involvement. SwitchReduce compressed wildcard identical action flows into fewer entries and maintains per-flow counters only at ingress switches. In the publication by Luo et al. [40] the authors developed FFTA and iFFTA, a pair of flow table aggregation and update schemes, to reduce the size of flow tables in SDN switches. FFTA used a modified binary search tree and ORTC techniques to achieve very fast offline non-prefix flow rule aggregation. iFFTA further incorporated online updates around 3 times faster than re-running FFTA, with minimal impact on compression ratio. The Espresso heuristic, presented in the publication by Braun et al. [41], proposed the compression of a minimum-size set of prefix-based match fields to achieve logic minimization within flow tables. DIFANE [31] employed decision trees to subdivide flow rules and assign them to corresponding switches, thereby circumventing controller bottlenecks.

Much attention has been paid to flow rule placement in addressing signaling overhead. Reactive and proactive categories of flow rule placement are explained in the following sections.

Reactive rule placement

A reactive approach to rule placement involves the placement of rules on flow-related events, such as table miss. As the OpenFlow specification outlines, the switch notifies the controller of its arrival for each new flow if there is no corresponding flow entry. Subsequently, the controller installs a new flow rule for the switch.

It should be noted that an increase in table misses leads to more frequent communication between switches and controllers for exchanging packet-in and packet-out messages. Moreover, setting up new flow rules takes considerable time, resulting in additional latency.

An example of reactive flow rule placement is presented in Fig. 2. Assuming that a flow enters the switch (Arrow 1), and its flow rule is not found in the flow table, the switch sends an OpenFlow message to the controller, and the controller sends the flow rule placement message to the switch (Arrow 2). After the switch deals with the flow (Arrow 3), it handles subsequent flows (Arrow 4,5).

Reactive flow rule placement necessitates continuous interaction between the switch and the controller. In the publication by Luo et al. [40], integer linear programming (ILP) and greedy heuristic algorithms were employed for flow rule placement. The controller put unused links into sleep mode to minimize energy consumption, while considering the constraints on link capacity and space of the table, keeping the rules. An energy-efficient traffic forwarding scheme was proposed in the publication by Giroire et al. [42], considering dynamic traffic in the

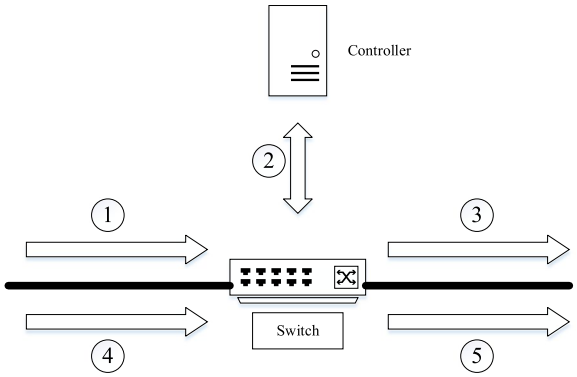


Fig. 2 The process of reactive flow rule placement

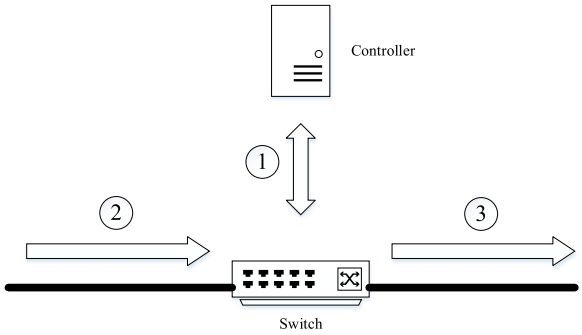


Fig. 3 The process of proactive flow rule placement

network. The traffic management policies minimizing unnecessary traffic are presented in the publication by Markiewicz et al. [43], and the routing strategy of the publication by Zhang et al. [44] was based on Open Shortest Path First (OSPF), utilizing the global view of the network. The authors in the publication [44] conducted a comprehensive examination of the performance of end-to-end delay in SDN networks with multiple nodes. They introduced a novel analytical framework and conducted measurements that exposed substantial delays when compared to conventional networks. This drew attention to critical factors that need to be taken into account when designing SDN switches and developing network algorithms to mitigate these delays.

It is important to note that reactive rule placement typically involves high communication load and latency for the devices involved. Additionally, due to the limitations in the computation speed of the controller, handling flow messages quickly is challenging. Therefore, it is typically employed for elephant flows.

Proactive rule placement

The proposed methodology involves pre-emptively placing rules before the arrival of new flows. It is based on predicting the usage of flow rules before their insertion into the flow table, as illustrated in Fig. 3. This approach can effectively reduce delays in setting up flow rules and minimize the total number of signaling messages.

Proactive flow rule placement is often integrated with access control, with ILP being utilized in [45] to design and analyze the dependency graph for rule placement in the switch. The graph is also helpful for predicting access control rules before the corresponding flow arrives [46–48]. DevoFlow [22] further reduced switch-controller interaction by detecting significant flows ahead of time. The switch only places flow rules associated with mice flow, without invoking the controller.

Figure 3 serves as an example of proactive flow rule placement, where flow rules are installed by the controller (Arrow 1), matched by the contents of a flow table (Arrow 2), and quickly forwarded without setup delays (Arrow 3).

In recent times, there has been an increased consideration of user mobility in managing SDN flow table due to the growing popularity of IoT. A flow rule placement scheme was proposed in the publication by Katta et al. [49] which made an improvement over the current method by taking into account the occupation time of a flow rule in a table while ensuring the processing of the flow with the local switch. In the publication by Li et al. [50], a dynamic environment allowed users to join and exit the network freely. In this publication, an online flow-based routing approach was presented, which permits the dynamic reconfiguration of the existing flows and the adaptation of the link rate, considering the user's demand and mobility. The publication by Wang et al. [51] introduced MoRule, a rule management scheme specifically designed for mobile users within Software Defined Networks (SDNs). Unlike existing approaches that heavily rely on static network topology, MoRule took into consideration the dynamic nature of mobile networks. It effectively utilized a low-complexity heuristic algorithm to optimize the placement of rules, while also ensuring that local switches efficiently handle mobile traffic above a pre-defined threshold. By capitalizing on the principles of software-defined networking (SDN), SDIV [51] proposed the segmentation of control and data planes, thereby providing a standardized means of configuration for a variety of switches. This innovative approach significantly improved service deployment and scalability. In order to tackle the limitations associated with Flow Tables in OpenFlow-enabled switches, Amokrane et al. [52] presented an approach for optimizing rules. This particular approach efficiently reduced rule complexity, while simultaneously preserving the performance of data transmission, as verified through a series of experiments. The publication by Liu et al. [53] presented a software-defined

IoT (SD-IoT) architecture for intelligent urban sensing, which customizes data acquisition, transmission, and processing for the user through well-defined APIs. Various applications coexisted in the common infrastructure to reduce overall maintenance costs. The publication by Anadiotis et al. [54] formulated an SDN-assisted system using MapReduce to support big data processing in wireless sensor networks (WSN). It allowed for the dynamic loading and implementation of user-specified maps and reduced energy consumption in the nodes. In the publication by Bera et al. [55], a mobility-aware adaptive flow rule placement, Mobi-Flow, was presented as a solution to the issue of user mobility. It consisted of two components: the path estimator and the flow manager. The path estimator predicted the future position of the user in the network, while the flow manager determined the activation of access points and placed the corresponding flow rules according to the result of the path estimator.

Next, the proposed MAPQ scheme is presented, further enhancing the Mobi-Flow approach using cost-sensitive AdaBoost and Q-learning. Frequent use of reactive flow rule placement is not beneficial for mice flow. The proposed MAPQ scheme adopts proactive flow rule placement to resolve the issue.

The proposed scheme

This section presents the proposed MAPQ scheme for mobility-aware flow rule placement with SDN. Then its function is analyzed, providing more space for high-level flow rules and assuring the efficiency of the flow table management.

Overview

As depicted in Fig. 4, the operation of MAPQ involves a tripartite process comprising flow rule classification, position prediction, and flow entry update. The first step of flow classification utilizes the cost-sensitive AdaBoost methodology to categorize the flows into two distinct types. The accuracy of this particular classification process holds substantial sway over the performance of the MAPQ scheme, wherein mice or inactive flow are excluded from consideration. The proactive rule placement policy allocates the remaining flows to the switches. The Q-learning model is leveraged to prognosticate the node's position relative to the corresponding flow entry, thereby facilitating the scheduling of rule placement in advance [23]. Subsequently, the controller updates the flow entries contingent upon the node position. The notations employed in this study are summarized in Table 2.

Flow classification

Flow classification is the first step of MAPQ, which classifies the flows into elephant flows or mice flows. The traditional machine learning algorithm uses a training dataset, $(X^{tra},Y^{tra})=\{(x_1,y_1),(x_2,y_2),\ldots,\ (x_i,y_i),\ \ldots,\ (x_n,y_n)\}$ collected from SDN, to generate a classifier, where (x_i,y_i) is the input features x_i with the target label y_i. Then it is employed to identify the category of the test dataset, (X^{tes},Y^{tes}). With AdaBoost [24], a trained classifier is used to differentiate two kinds of flows; elephant (active) flows and mice (inactive) flows. Usually, the distribution of the training dataset is the same as that of the test dataset. As for the training dataset of a different distribution, it is regarded as a redundant dataset. Unlike traditional machine learning algorithms, the cost-sensitive algorithm considers the minimization of the misclassification cost to construct a full-scale classifier.

Based on the cost-sensitive scheme framework, the improved AdaBoost algorithm, known as cost-sensitive AdaBoost, has been utilized in the context of MAPQ. AdaBoost is a conventional machine learning algorithm that comprises multiple weak classifiers. These classifiers are assigned weights based on the classification error, and a robust classifier is produced by aggregating the weak classifiers utilizing these weights. The operational sequence is outlined as follows.

1) Initialize the weight of each training datum, meaning one data point or one instance in your dataset, to be the same as $1/m$, where m refers to the count of individual instances, examples, or observations in the

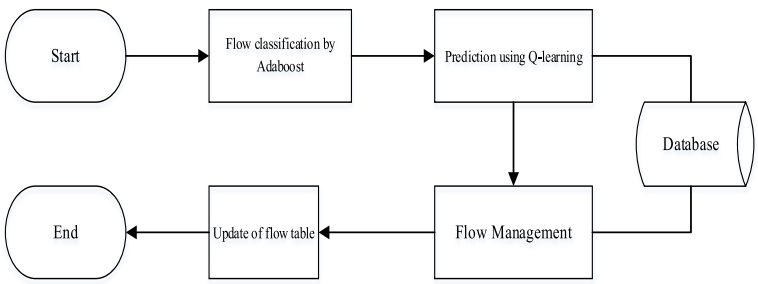


Fig. 4 The operation flow of the MAPQ scheme

Table 2 The list of notations

Parameter	Description
$w_t(i)$	weight of the i^{th} training example at the t^{th} iteration
$H(x)$	final strong classifier
acc	accuracy of classification
$c(y_i,y_j)$	tcost function, which assigns a cost to the event of predicting class y_j when the true class is y_i
N	total number of samples or instances in the dataset
p	number of negative samples
$L(x,y)$	real loss associated with a prediction for a given class y when the input is x
τ	index or identifier for the weak learners in the ensemble that the AdaBoost algorithm generates
a_τ	weight assigned to the τ-th classifier in the ensemble
$h_\tau(x)$	hypothesis or prediction made by the τ-th classifier for the sample x
L	set of all possible states in the environment
A	a set of all possible actions that the agent can take in a given state
R	reward received after transitioning from one state to another due to an action taken by the agent
P	probability of transitioning from one state to another
H	a dataset or a set of data points that encapsulate the historical movement of the end-device
l_i	i^{th} position of the device in a sequence of positions
t_i	arrival time of the device corresponding to i^{th} position
w_{ij}	weight of the visit from l_i to l_j
h_i	i^{th} basic classifier
P_{ij}	transition probability from l_i to l_j, $i \neq j$
α	learning rate
γ	discount factor
m	the total number of the training data
t	current iteration or round of the boosting process
$Q(s,a)$	expected cumulative reward for taking action a in state s
$Q'(s',a')$	estimated maximum reward for the next state s' over all possible actions a'.
$R(s,a)$	reward received after taking action a in state s

training dataset you are using to train your model, indicating that, at the start, each piece of training data is considered equally important in training the first weak classifier. Thus, the initial weights are

$$w_1 = w_2 = \cdots = w_m = \frac{1}{m} \tag{1}$$

2) Update the weights of the training data with n iterations.

A. Choose a weak classifier of the minimal error rate as the t^{th} basic classifier, h_t, and calculate its error rate.

$$\varepsilon_t = \Pr_{i\sim D_t}[h_t(x_i) \neq y_i]$$

$$= \sum_{i=1}^{m} w_{(t)}(i)\left[1 - I\left(h_t(x_i) = y_i\right)\right] \tag{2}$$

where $Pr_{i\sim Dt}$ The probability concerning the distribution D_t is contingent on the fact that the distribution D_t assigns a probability to each training example in proportion to its weight during iteration t. In the AdaBoost algorithm, the weights of the training examples are modified following each iteration in order to prioritize the examples that were previously misclassified. Consequently, the probability distribution D_t undergoes modifications during each iteration, which then influences the selection of the weak classifier that most effectively rectifies the prior errors. vAlso w_i^t denotes the weight for the t^{th} iteration and $I(\bullet)$ is the indicator function.

$$I(y = x) = \begin{cases} 1, y = x \\ 0, y \neq x \end{cases} \tag{3}$$

B. Calculate the weight of the basic classifier for the final ensemble.

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) \tag{4}$$

C. Update the weight of each training data.

$$w_{(t+1)}(i) = \frac{w_t(i)\exp\left(-\alpha_t y_i h_t(x_i)\right)}{Z_t} \tag{5}$$

where Z_t is the normalization factor.

$$Z_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)} \tag{6}$$

3) Combine the basic classifiers using the following equation into a strong classifier.

$$H(x) = \text{sign}\left(\sum_{t=1}^{T}\alpha_t h_t(x)\right) \tag{7}$$

The overall steps are summarized in Procedure 1.

Input: $(X^{tra},Y^{tra})=\{(x_1,y_1),(x_2,y_2),\ldots,(x_i,y_i),\ \ldots,(x_n,y_n)\}$ where $x_i \in X, y_i \in \{-1,1\}$.

Initialize: $w_1 = w_2 = \cdots = w_m = \frac{1}{m}$

1. FOR t=1 to n
2. Train weak learners using the current weight distribution, D_t
3. Obtain weak hypothesis h_t: $X \rightarrow \{-1,1\}$.
4. Select h_t of smallest weighted error:
$$\varepsilon_t = Pr_{i\sim D_t}[h_t(x_i) \neq y_i].$$
5. $$\alpha_t = \frac{1}{2} ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right).$$
6. Update for i=1,…,m:
$$w_{(t+1)}(i) = \frac{w_t(i)exp\left(-\alpha_t y_i h_t(x_i)\right)}{Z_t}$$
where Z_t is a normalization factor used to make $w_{(t+1)}$ a weight.
7. Compute the final hypothesis:
$$H(x) = sign\left(\sum_{t=1}^{T}\alpha_t\, h_t(x)\right)$$

Output: $H(x)$

Procedure 1 AdaBoost

AdaBoost is similar to traditional machine learning algorithms in adopting the hypothesis that the training and test dataset distributions are the same [24]. The cost-sensitive AdaBoost preserves AdaBoost for the training dataset of the same distribution, and a cost-sensitive function is applied to the training dataset of different distributions for flow classification. To differentiate the importance of the data, each data is associated with a cost:

Table 3 The combinations of prediction of the values

Prediction	Actual Negative	Actual Positive
Negative	$Cost_{TN}$:c(0,0)	$Cost_{FN}$:c(1,0)
Positive	$Cost_{FP}$:c(0,1)	$Cost_{TP}$:c(1,1)

the higher the cost, the higher the importance of correctly identifying the data. Let $\{(x_1,y_1,c_1),\ldots,(x_m,y_m,c_m)\}$ be a sequence of training dataset, where $c_i \in [0,+\infty)$ is an associated cost. $c(y_i,y_j)$ is the cost of predicting class y_j when the true class is y_i for a sample. Its value is decided below.

$$c(y_i,y_j) = \begin{cases} \frac{p}{N} \times acc & y_i \neq y_j \text{ and } y_i = -1 \\ \frac{1-p}{N} \times (1-acc) & y_i \neq y_j \text{ and } y_i = 1 \\ 0 & y_i = y_j \end{cases} \tag{8}$$

where $c(0,0)$ is the cost of correctly predicting negative value for actual negative value, $Cost_{TN}$. Four combinations exist regarding the correctness of the positive/negative value prediction as listed in Table 3. The update of the weight is done as follows.

$$w_{(t+1)}(i) = \frac{w_t(i)\,\exp\left(-\alpha_t y_i h_t(x_i) c\left(y_i, h_t\left(x_i\right)\right)\right)}{Z_t} \tag{9}$$

The final ensemble formula is

$$H(x) = \text{sign}\left[\sum_{y\in\{-1,1\}} L(x,y)\left(\sum_{\tau:h(x)=y} a_\tau h_\tau(x)\right)\right] \tag{10}$$

where $L(x,y)$ is the real loss of class_i. Here class_i is the actual class of x.

$$L(x,y_i) = \sum\nolimits_j p(j|x)c(y_i,y_j) \tag{11}$$

As the cost factor is involved in the proposed AdaBoost scheme, it can be regarded as a cost-sensitive boosting algorithm, summarized in Procedure 2. For the AdaBoost algorithm, selecting a suitable value for the weight update parameter is essential in transforming a weak classifier into a strong one. When the cost parameters are added to the weight updating formula of AdaBoost, the data distribution is affected by the parameter. The efficiency of AdaBoost is not guaranteed if the cost is considered without proper weight updates. The value of the weight update parameter needs to be decided to minimize the overall training error of the combined classifier. The process of flow classification is shown in Fig. 5.

Input: $(X^{tra},Y^{tra})=\{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}$ where $x_i \in X, y_i \in \{-1,1\}$.

Initialize: $w_1 = w_2 = \cdots = w_m = \frac{1}{m}$

1. FOR t=1 to n
2. Train weak learners using the current weight distribution, D_t.
3. Get weak hypothesis h_t: $X \rightarrow \{-1,1\}$.
4. Select h_t of the lowest weighted error:
$$\varepsilon_t = Pr_{i\sim D_t}[h_t(x_i) \neq y_i].$$
5. Choose
$$\alpha_t = \frac{1}{2} ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right).$$
6. Update for i=1,…,m:
$$w_{(t+1)}(i) = \frac{w_t(i)\exp(-\alpha_t y_i h_t(x_i) c(y_i, h_t(x_i)))}{Z_t}$$
Where Z_t is a normalization factor used to make $w_{(t+1)}$ a weight.
7. Compute the final hypothesis:
$$H(x) = sign\left[\sum_{y\in\{-1,1\}} L(x,y)\left(\sum_{\tau:h(x)=y} a_\tau h_\tau(x)\right)\right]$$

Output: $H(x)$

Procedure 2 Cost-sensitive AdaBoost

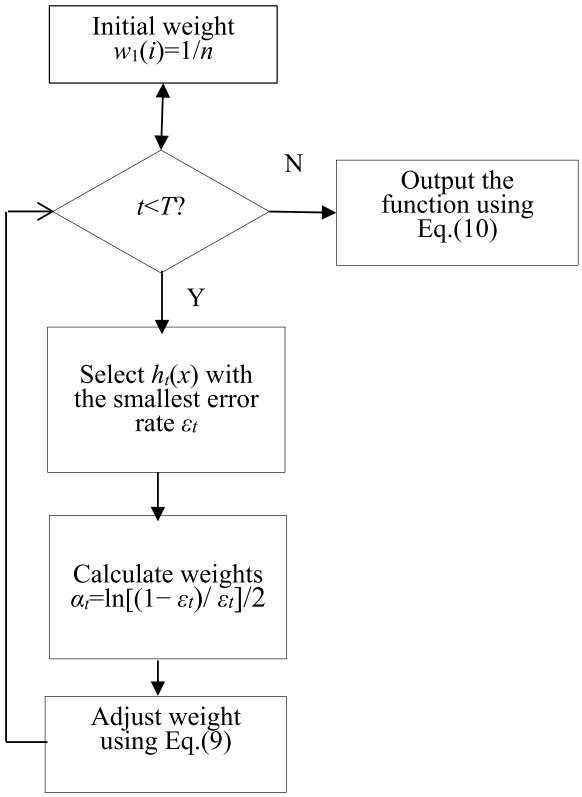


Fig. 5 The operation flow of cost-sensitive AdaBoost

Prediction of location

The QL algorithm is used to predict the future position of the end device. The QL model consists of 5-tuple <L,A,R,P,γ>, where L is a finite set of states, and A is a finite set of actions. The controller obtains feedback information which R denotes, and P is the transition probability between the states. The feedback for each end device is the same, and the discount factor γ is applied to the calculation of the reward value. First, two tuples, <H, P>, need to be elaborated.

- H: A set of data showing the movement history of an end-device with three tuples, <L,T,S>, where $L = \{l_1, l_2,\ldots,l_n\}$ is the location visited by the device, $T = \{t_1,t_2,\ldots,t_n\}$ is the arrival time, and $S = \{s_1,s_2,\ldots,s_n\}$ is the duration of stay at each location. n is the number of positions visited.
- P: Transition probability from one location to another as P_{ij} the transition probability from l_i to l_j, $i \neq j$.

The QL algorithm is typically composed of a series of actions carried out by agents and states, as well as a reward function and a transfer function. In the proposed QL model, the controller and SDN are considered the agent and environment, respectively, in which the agent operates. The agent initially assesses the state of the SDN and determines the appropriate course of action to interact with it. Once an action is taken, a reward is obtained and the environment transitions to a new state. Through ongoing interactions with the environment, the agent observes a sequence of state-action-reward events and assesses the reward in conjunction with the corresponding state and action pair. The ultimate objective is to obtain the expected maximum cumulative discounted reward, which the agent seeks to achieve by calculating the optimal action-value function.

The QL algorithm is a set of actions executed by agents and states, along with reward and transfer functions. Within the proposed QL model, the controller and SDN are identified as the agent and environment in which the agent conducts operations. At the outset, the agent evaluates the condition of the SDN and determines the most appropriate course of action for engaging with it. A reward is obtained upon taking action, and the environment moves to a new state. Through continual interactions with the environment, the agent observes a sequence of state-action-reward occurrences and evaluates the reward in conjunction with the corresponding state and action pair. The primary objective is attaining

the expected maximum cumulative discounted reward, which the agent accomplishes by computing the optimal action-value function.

$$Q(s,a) = (1-\alpha) \times Q(s,a) + \alpha\left[R(s,a) + \gamma \times \max Q'(s',a') - Q(s,a)\right] \tag{12}$$

where $Q(\bullet)$ is the expected reward, and α is the learning rate set to be the same value as QLAODV [50]. γ (>1) is the discount factor, determining the importance of the reward. In addition, $Q^{'}(s^{'}, a^{'})$ is the maximum reward given the new state, $s^{'}$, and all possible actions at $s^{'}$. The proposed scheme defines the state, action, and reward as follows.

Definition 7 (*State*). The state is deemed the location that the end device has formerly accessed, with each position signifying a semantic locale such as the office, home, station, etc. As previously mentioned, L denotes this.

Definition 8 (*Action*). An action possessing a particular state can be delineated as the duration during which the device remains stationary in its present position (state). This duration can be calibrated in seconds, ranging from 1-10 secs, and is therefore classified as an action space, A, comprising of the values {1, 2,...,10}. This is because the device's state is assessed at intervals of 10 secs. It should be noted that a decrease in the number of actions enhances the precision of prediction since the number of predicted actions is curtailed.

Definition 9 (*Reward*). The reward is given for a transition between two states with an action. The reward with s_i, R_i, is represented as follows.

$$R_i = \sum_{j=1}^{n} w_{ij} P_{ij} \tag{13}$$

where w_{ij} is the probability of a visit to l_j from l_i.

$$w_{ij} = P\left(l_{t+1} = l_j | L\right) = \frac{N\left(al_j, L\right)}{N(a, L)} \tag{14}$$

Here $a = l_1,\ldots,l_i$ is a sequence of i recent positions visited and $N(\bullet, L)$ is the number of occurrences of one sequence of the visited locations, L.

In Software-Defined Networking (SDN), the controller regularly verifies the location of the end device and its connectivity to a switch every 10 seconds. The OpenFlow protocol facilitates the collection of network information, thereby resulting in no additional burden in executing the suggested approach for acquiring network information.

Prediction operation

For predicting the following location, the Q-values are evaluated. Here the input is the current location, l_i, and the output is a vector, $Q(l_i) = (Q(l_i,a_1), Q(l_i,a_2),\ldots,Q(l_i,a_{10}))$, consisting of 10 estimated Q-values for each possible action. The following action a_i is chosen according to the vector $Q(l_i)$.

$$a_i = \underset{j\varepsilon[1,\ldots,10]}{\arg\max}\ Q\left(l_i, a_j\right) \tag{15}$$

Given the possibility of encountering a local optimum, the decision-making process employs the ε-greedy policy to determine the subsequent action [56]. Specifically, an action is randomly and uniformly selected with a small probability of ε, while the best action is chosen from the Q-table based on Eq. (15) with a probability of (1-ε). The controller selects the action with the highest value with a probability of (1-ε), and a random action with a probability of ε, thereby facilitating exploration of the environment with a low probability. It is noteworthy that the parameter, ε, is not constant, but rather varies based on the operational status of the scheme. Specifically, ε is set to $1/c$, where c denotes the number of successful predictions. The value of ε decreases gradually as the frequency of successful predictions increases, thereby promoting exploration during the initial stages of training, while the degree of exploitation in the later stages is contingent on experience.

Update operation

The model gets <L, T, S> of the end device delivered by the switch. Then the state of the end device is updated, and the corresponding reward for the action is computed. Lastly, the last state, current state, previous action, and its corresponding reward are logged in the training dataset for future use.

Train

The data are periodically sampled from the training dataset to perform experience replay, which utilizes previous data to remove the samples' associations. With the input consisting of past states, the Q-value of the pair of (s,a) is updated with the previous pair using the Bellman equation [57], while other Q-values are unchanged. The updated results are stored as the output. The key to using the Bellman equation is that if the Q-value of all the subsequent states is known, the optimal strategy is the action maximizing the expected value of $R(s, a) + \gamma \times max\ (Q^{'}(s^{'}, a^{'}) - Q(s, a))$.

The proposed QL algorithm for reinforcement learning exhibits a significant deviation from other such algorithms in that the subsequent state and corresponding reward are not immediately procured upon the selection of a specific action. Rather, they are obtained post the relocation of the end device by the controller. This implies that the parameters of QL do not undergo any alteration until the end device is relocated. Moreover, the choice of the next action is not contingent on the current action. The process of predicting the location of a device is explicated in the following manner.

Input: H and current location, l_i
Initialize: the number of predictions, t =0.
1. Decide optimal action according to the action function as shown in Eq. (15);
2. Take action based on the ε-greedy policy;
3. Determine the next state, l_j;
4. Compute reward
$$R_i = \sum_{j=1}^{n} w_{ij} P_{ij}$$
5. Update discrete Q-value
$$Q(s,a) = (1-\alpha) * Q(s,a) + \alpha[R(s,a) + \gamma * maxQ'(s',a') - Q(s,a)]$$
6. Formulate continuous actions through ε-greedy policy
7. t=t+1;
8. End while
Output: Sequence of predicted locations

Procedure 3: Prediction of location.

Rule placement

In an SDN-based IoT environment, the network comprises numerous end devices that are connected through distinct switches. These switches maintain flow tables that require consistent updates to optimize the hit ratio, considering the movement of the end devices. The proposed scheme known as MAPQ (mobility-aware flow rule placement with Q-learning) offers a resolution to this particular challenge by employing a proactive methodology to refresh the flow rules within the switches. In order to accomplish this, it is of utmost importance to monitor the future positions of end devices based on their past movements. This tracking mechanism establishes the foundation for effective flow rule placement subsequent to device movement within wireless environments.

Here is a comprehensive analysis of the flow rule placement process that transpires in response to device movement in such an environment:

i. Continuous Device Tracking: The SDN controller and associated network infrastructure are equipped with mechanisms to continuously monitor the movements of IoT end devices. These mechanisms can encompass various technologies such as GPS, signal strength monitoring, or triangulation techniques.
ii. Movement Prediction: Drawing upon historical movement patterns and real-time data collected, the SDN controller prognosticates the future location of each end device through the implementation of Q-learning algorithm.
iii. Proactive Rule Update: As an IoT device commences movement, the SDN controller proactively identifies the switch or switches that will be responsible for managing the device's traffic at its impending location. This is accomplished by considering factors such as proximity, the load on switches, and network traffic conditions.
iv. Flow Rule Placement: Once the future switch (or switches) has been ascertained, the SDN controller proceeds to update the flow table(s) of the corresponding switch(es) according to the flow classification using the cost-sensitive AdaBoost algorithm. This update entails the placement or modification of flow rules that delineate how traffic from the moving IoT device should be processed and forwarded.
v. Rule Removal at Previous Location: Concurrently, the SDN controller eliminates or adjusts the flow rules at the switch that corresponds to the device's previous location. This ensures the efficient utilization of network resources and the exclusion of unnecessary rules that are no longer pertinent to the switches.
vi. Seamless Connectivity: With the updated flow rules in effect, the IoT device can seamlessly maintain communication as it traverses the network. The flow rules at the current switch are consistently aligned with the device's anticipated or actual location, thereby optimizing the hit ratio and minimizing latency.
vii. Iterative Process: This process is iterative and dynamic, constantly adapting to changes in device movement patterns and network conditions. The SDN controller continuously refines its predictions and updates flow rules accordingly.

To summarize, the MAPQ scheme capitalizes on proactive flow rule placement based on predictive device movement tracking in wireless IoT environments. This approach ensures that the network remains responsive to the mobility of IoT devices, thereby upholding optimal connectivity and efficient resource utilization throughout the network.

Performance evaluation

In this section, a computer simulation is performed to assess the proposed approach and contrast it with the existing methods, namely the standard hard time-out scheme and Mobi-Flow scheme [51] which employs a location prediction mechanism in conjunction with an adapted time-out.

Simulation setting

The simulation is implemented on Intel Core i7 processor, 2.50GHz PC with 16GB RAM, and Matlab R2015a. The prediction accuracy of a scheme, *Acc*, indicates the rate of correct prediction of the flow entry having the most negligible matching probability. When the hard time-out is up, the predicted flow entry is checked whether it has the smallest matching number. Then the number of successful predictions, n_s, is incremented by 1. After the entire operation is over, the accuracy is calculated as:

$$Acc = \frac{n_s}{n_e} \tag{16}$$

where n_e is the number of evicted flow entries. Table 4 lists the simulation parameters. The flows are generated according to an exponential distribution with $\lambda = 1$, and the contents of the flow entries are decided randomly.

A virtual network of tree topology of 25 switches and 200 hosts of Fig. 6 is constructed using NS3 with the following elements: Open vSwitch, open daylight controller, and end-host nodes with the service provider (SP) enabled. The host nodes, the switch, and the controller are running on Ubuntu 16.04 LTS, and all the switches are connected to a controller.

Table 4 The simulation parameters

Parameter	Value
Number of switches	3
Number of flow entries per table	100 ~ 200
Upper bound on controller-to-switch delay	3.043 ms
Upper bound on end-to-end delay	0.341 ms
Average packet arrival rate per switch	0.005 ~ 0.025mpps (million packets per second)
Average packet service rate per switch	0.30mpps
Flow-table lookup time	33.333 µsec

Experiment results

Suppose that f and g denote the number of incoming flows and flow entries of a flow table, respectively. The performance metrics adopted in the simulation are *Precision*, *Recall*, *F1-score*, and *Garbage rate* (*Grate*). The number of data correctly classified as positive while they are positive is defined as true positive (*TP*). The number of data mistakenly classified as a positive class for a negative one is defined as false positives (*FP*). *TN* and *FN* are defined similarly. *Precision*, obtained by Eq. (17), is the ratio of True Positives to all the Positives. *Recall*, defined as Eq. (18), represents the portion of correctly classified positive data from the positive dataset. *F1_score* of Eq. (19) is a parameter obtained by the harmonic mean of *precision* and *recall*. *FP* and *FN* deteriorate the performance of a classifier. *Grate*, defined as Eq. (20), is the portion of false classification out of entire classifications used to show the overall inaccuracy.

$$Precision = TP/(TP + FP) \tag{17}$$

$$Recall = TP/(TP + FN) \tag{18}$$

$$F1 - score = 2 \bullet Precision \bullet Recall/(Precision + Recall) \tag{19}$$

$$Grate = (FP + FN)/(TP + TN + FP + FN) \tag{20}$$

Table 5 compares the *Precision* and *Grate* of the classification of the proposed cost-sensitive AdaBoost approach and that of the traditional AdaBoost. Our analysis reveals that the cost-sensitive AdaBoost consistently outperforms AdaBoost in Precision and Grate. AdaBoost cannot filter bursty error readings, resulting in lower precision when error cost is not considered. Our simulation results demonstrate that incorporating error cost significantly enhances classification precision.

The comparison of table misses for the various schemes is presented in Fig. 7, with (8000,100) for (f,g). T_I and T_H are 18 s and 30s, respectively. Notably, the number of table misses exhibits an increase when f is increased from 8000 to 16,000 while g remains unchanged, as evidenced in the middle of the figure. However, it is worth mentioning that the number of table misses associated with the proposed scheme is considerably lower than that of other schemes. Furthermore, upon increasing the number of flow entries to 200, there is a significant reduction in table misses. It is noteworthy that the proposed MAPQ scheme substantially reduces the number of table misses compared to the other schemes, owing to

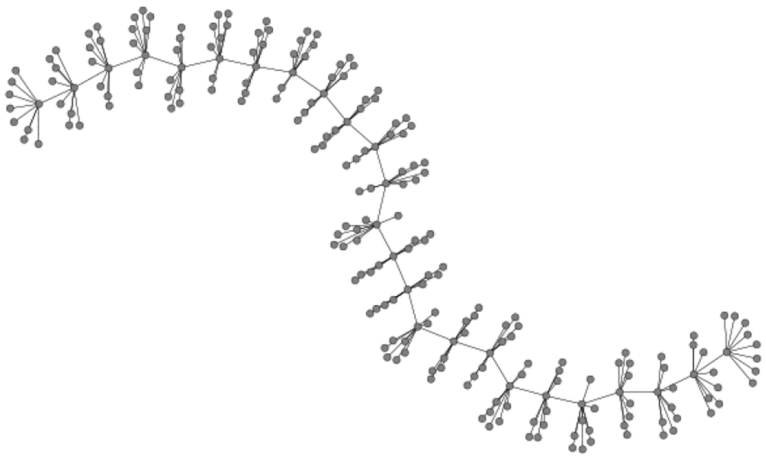


Fig. 6 The simulated topology

Table 5 The comparison of the metrics

Method	Recall	Precision	F1-score	Grate
Adaboost	0.74	0.91	0.81	0.05
Cost-sensitive Adaboost	0.56	0.95	0.71	0.03

its ability to predict the locations of the end devices with great accuracy.

The impact of T_H and T_I on the frequency of table misses is analyzed in Fig. 8, utilizing a sample size of 300 matches and employing the (8000,100) configuration. The data depicted in the figures reveal that the proposed scheme demonstrates a significant superiority over alternative schemes, irrespective of the values assigned to T_H and T_I. This suggests that factors such as network speed, packet processing rate, or collision/hashing efficiency in table management play a dominant role in the performance metrics. In such scenarios, even if the timeouts are adjusted, these other factors may have a greater impact, resulting in a consistent number of table misses.

Subsequently, to assess the efficacy of the flow table, an analysis is conducted on the number of obsolete flow rules and the volume of packets dispatched to the controller. It is worth noting that the (f,g) values employed correspond to those illustrated in Fig. 7, whereby T_I and T_H are 18 s and 30s, respectively. Remarkably, the

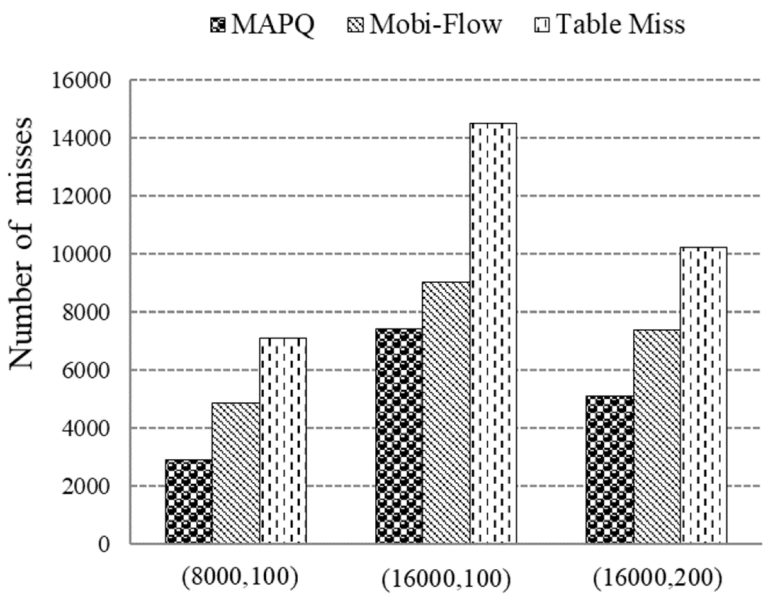
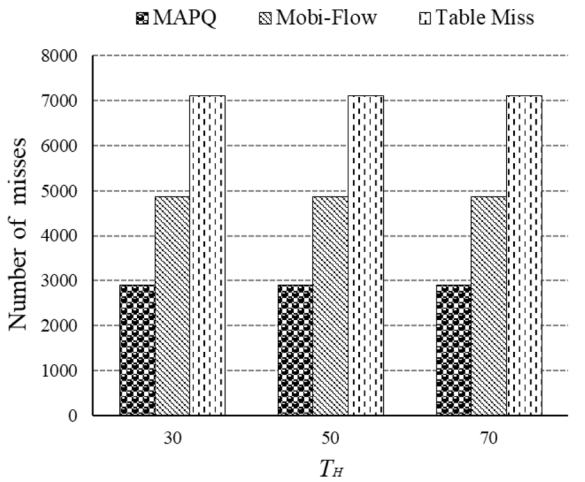
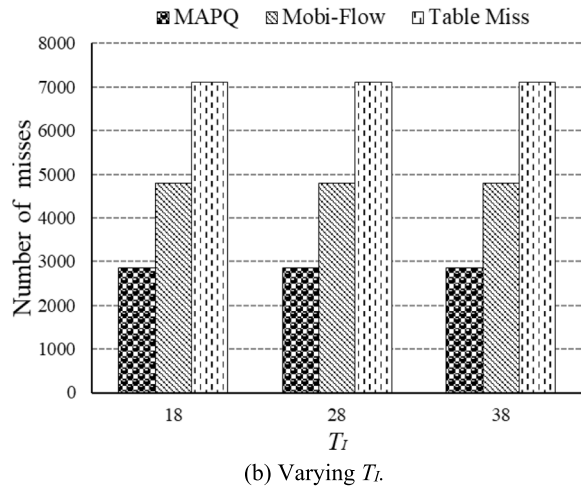


Fig. 7 The comparison of the number of tables misses with varying (f,g). (a) Varying T_H. (b) Varying T_I



(a) Varying T_H.



(b) Varying T_I.

Fig. 8 The comparison of the number of tables misses with varying T_H and T_I

proposed approach consistently outperforms other methods. Specifically, when utilizing (16,000,100), the number of inactive flow rules is inferior to that of (8000,100), given that the former facilitates support for additional flows with the same flow table size. However, when the number of flow entries is raised to 200, the number escalates due to an increase in unmatched flow entries. As illustrated in Fig. 9, the proposed mechanism consistently results in a notably reduced number of packets transmitted to the controller.

Figure 10 presents a comparison of the table occupancy rates among various schemes. This rate is the ratio of occupied flow entries to the total number of entries in the flow table. Notably, the table occupancy rate decreases when f increases from 8000 to 16,000 while g remains constant. This phenomenon can be attributed to the utilization of more flow entries with an increase in incoming flows. However, the proposed scheme exhibits significantly lower table occupancy rates compared to other schemes, owing to a reduction in the number of idle flow entries. Notably, when the number of flow entries increases to 200, the rate also increases, albeit being smaller than that of (8000,100), since more unmatched flow entries exist. Note that the proposed scheme shows a much lower occupancy rate than the other schemes, regardless of the operational condition, due to efficient flow classification and hybrid flow rule placement with the flow table.

Table 6 compares the prediction accuracy of the MAPQ scheme and Mobi-Flow. The (f,g) values used for the comparison are (8000,100) and T_I is set at 30 seconds. It is important to note that Mobi-Flow employs an order-k Markov predictor, whereas MAPQ utilizes Q-learning for flow prediction. The table demonstrates that the proposed scheme consistently achieves greater accuracy than Mobi-Flow across all tested scenarios.

Finally, the performance of the schemes is investigated in terms of average transmission delay and the ratio of packet drop in the switch with 300 matchings (8000,100). The comparison summary is presented in Table 7. The findings indicate that the proposed scheme is superior as it accurately predicts the end-device location, leading to reduced table misses. This reduction is achieved by minimizing the unnecessary flow entry placement, which enables the overhead of flow setup. As a result, the proposed scheme necessitates the smallest delay and packet drop rate, which, in turn, ensures the maintenance of free space in the SDN buffers (such as the datapath buffer that retains unmatched packets) while keeping the current connections undisturbed.

Conclusion

This paper presents an innovative approach to placing flow rules within the context of Software-Defined Networking (SDN) to support Internet of Things (IoT) applications. The proposed method leverages the Q-learning algorithm to anticipate the location of end devices, enabling proactive placement of flow entries. Using the forecasted results, the controller adjusts flow rules accordingly. Additionally, the cost-sensitive AdaBoost algorithm is applied to filter out small or rarely encountered flows. A computer simulation demonstrates that this approach significantly improves match probability and reduces the number of table misses compared to existing methods.

Future research will refine this approach by incorporating additional factors that influence prediction accuracy, such as idle time-out. The paper emphasizes proactive flow rule placement based on predicted

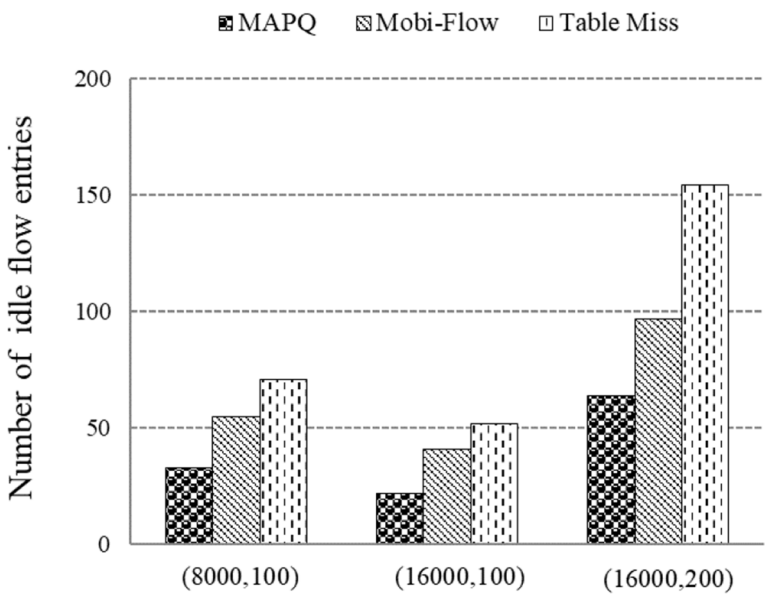


Fig. 9 The comparison of the number of idle flow entries

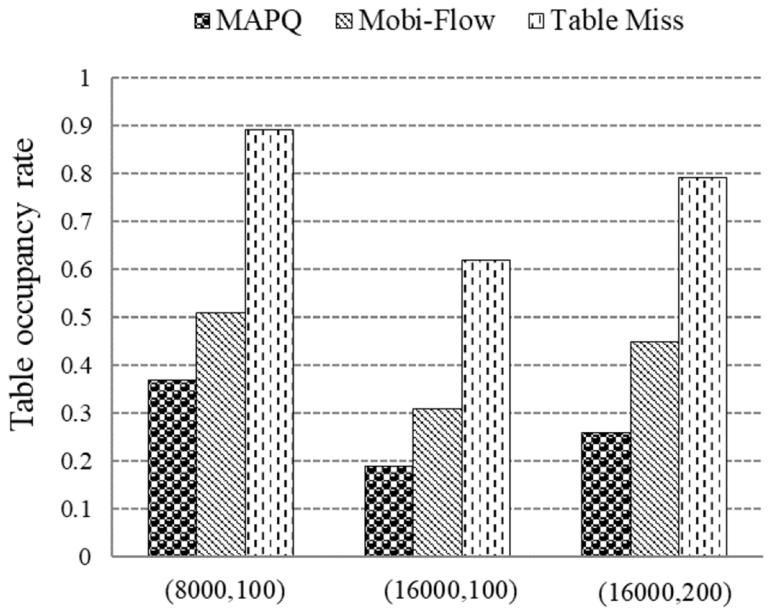


Fig. 10 The comparison of table occupancy rates

Table 6 The comparison of prediction accuracy

Method	1	2	3
MAPQ	0.815	0.832	0.809
Mobi-Flow	0.684	0.732	0.726

Table 7 The comparison of average delay and rate of packet drop

Scheme	MAPQ	Mobi-Flow	Table Miss
Average delay (ms)	3.29	4.21	7.35
Rate of packet drop	0.389	0.602	0.764

end-device locations and intends to explore a hybrid flow rule placement approach to better adapt to changing network conditions. Furthermore, an analytical model will be developed to achieve optimal designs for specific scenarios by establishing relationships between the considered factors and desired performance metrics.

Acknowledgments
Thanks to Dr. Hanyao Huang of Nanyang Institute of Technology for his help in our work.

Authors' contributions
Methodology: Gan Huang; Resources: Hanyao Huang and Ihsan Ullah; Software: Ihsan Ullah; Supervision: Kyung Tae Kim; Writing original draft: Gan Huang; Writing review editing Hanyao Huang, Kyung Tae Kim; All authors read and approved the final manuscript.

Authors' information
Gan Huang received the B.S. degree in Electronic Information Engineering from Chuzhou University, China, in 2012, the M.S. degree in Computer Science from Anhui Polytechnic University, China, in 2016, and the Ph.D. degree in Computer Engineering from Sungkyunkwan University, South Korea, in 2021. From 2021 to 2022, he was a postdoctoral researcher at Sabanci University, Turkey. He is currently a lecturer with the School of Mathematics and Computer Science, Zhejiang A&F University, China. His research interests include Software-defined networking (SDN), Quality of Experience (QoE), Ubiquitous and Distributed Computing, Cloud and Edge Computing, Mobile Computing, Wireless Sensor Networks, IoT, Machine Learning, Computer Networking, and Network Security.
Ihsan Ullah received the B.S. and M.S. degrees in computer science from the University of Peshawar, Pakistan, in 2001 and 2004, respectively, and the Ph.D. degree in computer engineering from Sungkyunkwan University, Suwon, South Korea, in 2019. From September 2019 to August 2020, he was a Postdoctoral Research Fellow with the Ubiquitous Computing Technology Research Institute (UTRI), Sungkyunkwan University. Since 2020, he has been a Research Professor with the School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan, South Korea. His research interests include Data aggregation, Data fusion, Virtual network embedding, Network slicing (5G), IoT (Internet of Things), Artificial Intelligence, Deep Reinforcement Learning, Cloud computing, and Networking.
Hanyao Huang received the B.S. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2015, and the Ph.D. degree from Sungkyunkwan University, Korea, in 2022.
He is a lecturer with the Department of Computer and Software Engineering, Nanyang Institute of Technology, Nanyang, China. His current research interests include Wireless Networks, Internet of Things Technology, and Machine Learning.
Kyung Tae Kim received the Ph.D. degree from College of Information and Communication Engineering at Sungkyunkwan University, Korea in 2013. He is currently a research professor at the College of Computing and Informatics at Sungkyunkwan University, Korea. His current research interests include Edge computing, Artificial Intelligence Systems, and Internet of Things technology.

Funding
This research was supported by Basic Science Research Programs through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2022R1I1A1A01053800).

Availability of data and materials
The data used to support the findings of this study are available from the corresponding author upon request.

Declarations

Ethics approval and consent to participate
No ethical approval is required, and the authors express their consent to participate in the paper.

Consent for publication
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 28 June 2023 Accepted: 3 January 2024
Published online: 25 January 2024

References

1. Bannour SS, Mellouk A (2018) Distributed SDN Control: Survey, Taxonomy, and Challenges. IEEE Commun Surv Tutor 20(1):333–354
2. Basit A, Qaisar S, Rasool SH, Ali M (2017) SDN Orchestration for Next Generation Inter-Networking: A Multipath Forwarding Approach. Ieee Access 5:13077–13089
3. Adami B, Martini A, Sgambelluri L, Donatini M, Gharbaoui PC, Giordano S (2017) An SDN orchestrator for cloud data center: system design and experimental evaluation. Trans Emerg Telecommun Technol 28(11):e3172
4. Aguado A, López V, Marhuenda J, de Dios ÓG, Fernández-Palacios JP (2015) ABNO: A feasible SDN approach for multivendor IP and optical networks. J Opt Commun Netw 7(2):A356–A362
5. Bastam M, Sabaei M, Yousefpour R (2018) A scalable traffic engineering technique in an SDN-based data center network. Trans Emerg Telecommun Technol 29(2):e3268
6. Abdelmoniem M, Bensaou B, Abu AJ (2018) Mitigating incast-TCP congestion in data centers with SDN. Ann Telecommun 73(3–4):263–277
7. Chekired LK, Mouftah HT (2018) Decentralized cloud-SDN architecture in smart grid: A dynamic pricing model. IEEE Trans Industr Inform 14(3):1220–1231
8. Chen J, Chen JB, Ling JC, Zhou JL, Zhang W (2018) Link failure recovery in SDN: high efficiency, strong scalability and wide applicability. J Circuits, Syst Comput 27(6):1850087
9. Chen N, Wang M, Zhang N, Shen XM, Zhao DM (2017) SDN-based framework for the PEV integrated smart grid. IEEE Netw 31(2):14–21
10. Babangida I, Bakar KBA (2023) Managing smart technologies with software-defined networks for routing and security challenges: A survey. Comput Syst Sci Eng 47(2):1839–1879
11. Al-Rubaye S, Aulin J (2017) Grid modernization enabled by Sdn controllers: leveraging interoperability for accessing unlicensed band. IEEE Wirel Commun 24(5):60–67
12. Kim ED, Choi Y, Lee SI, Kim HJ (2017) Enhanced Flow Table Management Scheme With an LRU-Based Caching Algorithm for SDN. Ieee Access 5:25555–25564
13. Metter M, Seufert F, Wamser TZ, Tran-Gia P (2017) Analytical model for SDN signaling traffic and flow table occupancy and its application for various types of traffic. IEEE Trans Netw Serv Manag 14(3):603–615
14. Qiu XF, Zhang K, Ren QZ (2017) Global flow table: A convincing mechanism for security operations in SDN. Comput Netw 120:56–70
15. Luo SX, Yu HF, Li LM (2015) Practical flow table aggregation in SDN. Comput Netw 92:72–88
16. Dargahi T, Caponi A, Ambrosin M, Bianchi G, Conti M (2017) A survey on the security of Stateful SDN data Planes. IEEE Commun Surv Tutor 19(3):1701–1725
17. Wei YK, Zhang XN, Xie L, Leng SP (2016) Energy-aware traffic engineering in hybrid SDN/IP backbone networks. J Commun Netw 18(4):559–566
18. Qi QL, Wang WD, Gong XY, Que XR (2017) Rules placement with delay guarantee in combined SDN forwarding element. KSII Trans Internet Inf Syst 11(6):2870–2888
19. Nguyen X-N, Saucez D, Barakat C, Turletti T (2016) Rules placement problem in openflow networks: a survey. IEEE Commun Surv Tutor 18(2):1273–1286
20. Isyaku B, Zahid MSM, Kamat MB, Bakar KA, Ghaleb FA (2020) Software defined networking flow table management of openflow switches performance and security challenges: A survey. Future Internet 12(9):147
21. Jordehi R (2015) Enhanced leader PSO (ELPSO): a new PSO variant for solving global optimisation problems. Appl Soft Comput 26:401–417

22. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks." pp. 254–265
23. Wang L, Mao W, Zhao J, Yuedong X (2021) DDQP: A double deep Q-learning approach to online fault-tolerant SFC placement. IEEE Trans Netw Serv Manag 18(1):118–132
24. Jiang X, Yuan X, Ke W, Zhang Y, Zhu Q-X, He Y-L (2022) An imbalanced multifault diagnosis method based on bias weights AdaBoost. IEEE Trans Instrum Meas 71:1–8
25. Estan C, Varghese G (2003) New directions in traffic measurement and accounting: focusing on the elephants, ignoring the mice. ACM Trans Comput Syst 21(3):270–313
26. Gude N, Koponen T, Pettit J, Pfaff B, Casado M, McKeown N, Shenker S (2008) NOX: towards an operating system for networks. ACM SIGCOMM Comput Commun Rev 38(3):105–110
27. Zhang L, Lin R, Shizhong X, Wang S (2014) AHTM: achieving efficient flow table utilization in software defined networks. In: 2014 IEEE Global Communications Conference. IEEE, pp 1897–1902
28. Zhang L, Wang S, Shizhong X, Lin R, Hongfang Y (2015) TimeoutX: an adaptive flow table management method in software defined networks. In: 2015 IEEE Global Communications Conference (GLOBECOM). IEEE, pp 1–6
29. Vishnoi, Anilkumar, Rishabh Poddar, Vijay Mann, and Suparna Bhattacharya (2014) "Effective switch memory management in OpenFlow networks." In Proceedings of the 8th ACM international conference on distributed event-based systems, pp. 177–188
30. Babangida I, Kamat MB, Bakar K b A, Zahid MSM, Ghaleb FA (2020) IHTA: dynamic idle-hard timeout allocation algorithm based OpenFlow switch. In: 2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE). IEEE, pp 170–175
31. Yu M, Rexford J, Freedman MJ, Wang J (2010) Scalable flow-based networking with DIFANE. ACM SIGCOMM Comput Commun Rev 40(4):351–362
32. Challa R, Lee Y, Choo H (2016) Intelligent eviction strategy for efficient flow table management in openflow switches. In: 2016 IEEE NetSoft Conference and Workshops (NetSoft). IEEE, pp 312–318
33. Kannan K, Banerjee S (2014) Flowmaster: Early eviction of dead flow on sdn switches. In: Distributed Computing and Networking: 15th International Conference, ICDCN 2014, Coimbatore, India, January 4–7, 2014. Proceedings 15. Springer Berlin Heidelberg, pp 484–498
34. Huang G, Youn HY (2020) Management of Flow Table of SDN for Proactive Eviction Using Fuzzy Logic. Front Comput Sci 14(4):1–10
35. Huang G, Youn HY (2020) Proactive eviction of flow entry for SDN based on hidden Markov model. Front Comput Sci 14:1–10
36. Draves, Richard P., Christopher King, Srinivasan Venkatachary, and Brian D. Zill (1999) "Constructing optimal IP routing tables." In IEEE INFOCOM'99. Conference on computer communications. Proceedings. Eighteenth annual joint conference of the IEEE computer and communications societies. The future is now (cat. No. 99CH36320), vol. 1, pp. 88–97. IEEE
37. Liu, Yaoqing, Xin Zhao, Kyuhan Nam, Lan Wang, and Beichuan Zhang (2010) "Incremental forwarding table aggregation." In 2010 IEEE Global Telecommunications Conference GLOBECOM 2010, pp. 1–6. IEEE
38. Kanizo Y, Hay D, Keslassy I (2013) Palette: distributing tables in software-defined networks. In: 2013 proceedings IEEE INFOCOM. IEEE, pp 545–549
39. Iyer AS, Mann V, Samineni NR (2013) SwitchReduce: reducing switch state and controller involvement in OpenFlow networks. In: 2013 IFIP networking conference. IEEE, pp 1–9
40. Luo S, Hongfang Y (2014) Fast incremental flow table aggregation in SDN. In: 2014 23rd international conference on computer communication and networks (ICCCN). IEEE, pp 1–8
41. Braun W, Menth M (2014) Wildcard compression of inter-domain routing tables for openflow-based software-defined networking. In: 2014 third european workshop on software defined networks. IEEE, pp 25–30
42. Giroire F, Moulierac J, Phan TK (2014) Optimizing rule placement in software-defined networks for energy-aware routing. In: 2014 IEEE global communications conference. IEEE, pp 2523–2529
43. Markiewicz A, Tran PN, Timm-Giel A (2014) Energy consumption optimization for software defined networks considering dynamic traffic. In: 2014 IEEE 3rd international conference on cloud networking (CloudNet). IEEE, pp 155–160
44. Zhang T, Liu B (2019) Exposing end-to-end delay in software-defined networking. Int J Reconfigurable Comput 2019
45. Vawter I, Pan D, Ma W (2014) Emulation performance study of traffic-aware policy enforcement in software defined networks. In: 2014 IEEE 11th international conference on Mobile ad hoc and sensor systems. IEEE, pp 775–780
46. Caria M, Jukan A, Hoffmann M (2016) SDN partitioning: A centralized control plane for distributed routing protocols. IEEE Trans Netw Serv Manag 13(3):381–393
47. Zhang S, Ivancic F, Lumezanu C, Yuan Y, Gupta A, Malik S (2014) An adaptable rule placement for software-defined networks. In: 2014 44th annual IEEE/IFIP international conference on dependable systems and networks. IEEE, pp 88–99
48. Kang, Nanxi, Zhenming Liu, Jennifer Rexford, and David Walker (2013) "Optimizing the" one big switch" abstraction in software-defined networks." In Proceedings of the ninth ACM conference on Emerging networking experiments and technologies, pp. 13–24
49. Katta, Naga, Omid Alipourfard, Jennifer Rexford, and David Walker (2014) "Infinite cacheflow in software-defined networks." In Proceedings of the third workshop on Hot topics in software defined networking, pp. 175–180. 2014. Optimized rule placement for mobile users in SDN-enabled access networks." In 2014 IEEE Global Communications Conference, pp. 4953–4958. IEEE
50. Li H, Li P, Guo S (2014) MoRule: optimized rule placement for mobile users in SDN-enabled access networks. In: 2014 IEEE global communications conference. IEEE, pp 4953–4958
51. Wang X, Wang C, Jiang C, Yang L, Li Z, Zhou X (2015) "Rule optimization for real-time query service in software-defined internet of vehicles." arXiv preprint arXiv:1503.05646
52. Amokrane A, Langar R, Boutaba R, Pujolle G (2015) Flow-based management for energy efficient campus networks. IEEE Trans Netw Serv Manag 12(4):565–579
53. Liu J, Li Y, Chen M, Dong W, Jin D (2015) Software-defined internet of things for smart urban sensing. IEEE Commun Mag 53(9):55–63
54. Anadiotis A-CG, Morabito G, Palazzo S (2015) An SDN-assisted framework for optimal deployment of MapReduce functions in WSNs. IEEE Trans Mob Comput 15(9):2165–2178
55. Bera S, Misra S, Obaidat MS (2018) Mobi-flow: mobility-aware adaptive flow-rule placement in software-defined access network. IEEE Trans Mob Comput 18(8):1831–1842
56. Tripathi S, Pandey OJ, Hegde RM (2023) "An optimal reflective elements grouping model for RIS-assisted IoT networks using Q-learning". IEEE Transactions on Circuits and Systems II: Express Briefs 70(8):3214–3218
57. Moon J (2020) Generalized risk-sensitive optimal control and Hamilton–Jacobi–bellman equation. IEEE Trans Autom Control 66(5):2319–2325

Publisher's Note