Baghdad Science Journal

Volume 23 | Issue 6 Article 28

6-26-2026

Predictive Mobility-Aware Analysis of Controller Overhead in SDN-Enabled IoT

Abdulrahman Saidu
Faculty of Computing, Univeristi Teknologi Malaysia, Johor, MALAYSIA, 81310 Johor Bahru, Johor, Malaysia AND Department of Computer Science, School of Science and Technology Federal Polytechnic Bali, Taraba State, Nigeria, abdulrahmansaidu@gmail.com

Kamalrulnizam Bin Abu Bakar
Faculty of Computing, Univeristi Teknologi Malaysia, Johor, MALAYSIA, 81310 Johor Bahru, Johor, Malaysia, knizam@utm.my

Babangida Isyaku
Department of Computer Science, Faculty of Computing and Information Technology, Sule Lamido University, P.M.B. 048, K/hausa, Jigawa State, Nigeria, Babangida.isyaku@slu.edu.ng

Muhammad Nura Yusuf
Department of Computer Science, Faculty of Computer and Information Technology, Abubakar Tafawa Balewa University, Bauchi PMB 0284, Nigeria, ymnura@atbu.edu.ng

Farkhana Binti Muchtar
Faculty of Computing, Univeristi Teknologi Malaysia, Johor, MALAYSIA, 81310 Johor Bahru, Johor, Malaysia, farkhana@utm.my

How to Cite this Article

Saidu, Abdulrahman; Bakar, Kamalrulnizam Bin Abu; Isyaku, Babangida; Yusuf, Muhammad Nura; and Muchtar, Farkhana Binti (2026) "Predictive Mobility-Aware Analysis of Controller Overhead in SDN-Enabled IoT," *Baghdad Science Journal*: Vol. 23: Iss. 6, Article 28.
DOI: https://doi.org/10.21123/2411-7986.5342

BAGHDAD SCIENCE JOURNAL 2026;23(6):2336–2351

RESEARCH ARTICLE

Predictive Mobility-Aware Analysis of Controller Overhead in SDN-Enabled IoT

Abdulrahman Saidu [1,2,*], **Kamalrulnizam Bin Abu Bakar** [1], **Babangida Isyaku** [3], **Muhammad Nura Yusuf** [4], **Farkhana Binti Muchtar** [1]

[1] Faculty of Computing, Univeristi Teknologi Malaysia, Johor, MALAYSIA, 81310 Johor Bahru, Johor, Malaysia
[2] Department of Computer Science, School of Science and Technology Federal Polytechnic Bali, Taraba State, Nigeria
[3] Department of Computer Science, Faculty of Computing and Information Technology, Sule Lamido University, P.M.B. 048, K/hausa, Jigawa State, Nigeria
[4] Department of Computer Science, Faculty of Computer and Information Technology, Abubakar Tafawa Balewa University, Bauchi PMB 0284, Nigeria

ABSTRACT

The rapid growth and increasing mobility of Internet of Things (IoT) mobile devices significantly intensify flow setup requests in Software-Defined Networking (SDN). Frequent movement of mobile devices across access points triggers repeated flow rule installations in data-plane switches, leading to excessive controller overhead under limited flow-table memory capacity. Inaccurate mobility prediction in existing approaches further aggravates this challenge by causing unnecessary rule placement and redundant flow setup requests. This study investigates the impact of mobility-aware prediction on controller overhead with experimental scenarios using different groups of mobile IoT devices, while considering controller CPU and memory utilization. A proactive mobility prediction model is employed to reduce reactive controller interactions. The proposed prediction model demonstrated approximately 100% accuracy compared to the benchmark model, significantly reducing unnecessary flow setup requests and minimising controller overhead. Furthermore, sensitivity analysis reveals that lower prediction error reduces controller load, whereas higher prediction error leads to increased controller overhead, highlighting the importance of accurate mobility prediction for scalable SDN-enabled IoT networks.

Keywords: Controller, IoT, Mobility, Prediction, SDN-enabled

Introduction

The application of the Internet of Things (IoT) has revolutionized the process of accessing and exchanging information in a globalized system. This has supported modern technological advancement with associated challenges. The demand for IoT applications, particularly with mobile devices, has expanded the growth of the internet into heterogeneous and large network environments. In,[1] the radio network evolves from 4G to 5G to accommodate the rapid increase in traffic generated by mobile devices; the wireless infrastructure is becoming more complex, making it challenging to manage the entire system. To address this challenge Software Defined-enabled Internet of Things (SDN-enabled IoT) is introduced. Software Defined Networking (SDN) is a flow-based network that separates the control plane from the data plane. The control plane (network control logic) uses the controller to install flow rules into the flow-table of a switch to obtain information about the entire network for effective control and management. This provides the controller with global knowledge and constantly handles the incoming flows into the

Received 14 May 2025; revised 25 February 2026; accepted 7 March 2026.
Available online 26 June 2026

* Corresponding author.
E-mail addresses: abdulrahmansaidu@gmail.com (A. Saidu), knizam@utm.my (K. B. Abu Bakar), Babangida.isyaku@slu.edu.ng (B. Isyaku), ymnura@atbu.edu.ng (M. N. Yusuf), farkhana@utm.my (F. B. Muchtar).

https://doi.org/10.21123/2411-7986.5342

network. Similarly, in the perception layer of the SDN-enabled IoT data plane (forwarding plane), some devices are stationary (immobile). In contrast, others are mobile, which requires constant monitoring of the device's location via access points (APs) for information exchange. Mobile nodes are connected to different access points (APs) and move from one location to another at different times (from time t_1 to time t_2). They generate traffic flows into the network via APs, and the flow table of a switch is responsible for storing the flows. Unfortunately, the switch has limited flow-table memory to accommodate large flow entries because it is power-hungry and costly.[2] Therefore, accurately monitoring and predicting the location of mobile devices is challenging, leading to unnecessary flow rule placement. Accordingly, several machine-learning approaches were conducted to address mobility prediction.[3–6] However, mobile devices can generate a large number of traffic flows, requiring different QoS, generating frequent flow setup requests between the controller and the switches. In this regard, the controller must provide the corresponding flow rule for each traffic flow to avoid table miss entries. Table miss always triggers flow setup requests, which introduces controller overhead. Therefore, each flow setup request requires the controller to update the network state. Notwithstanding, as the number of mobile devices increases, the flow setup increases. This process creates room for rule placement problems in accessing the network. To address this challenge, several mobility-aware approaches have been carried out by.[7,8]

The existing solutions can be classified into two parts to address the rule placement problem. Some researchers adopted machine learning, while others used mobility-aware approaches. However, both approaches experienced inaccurate prediction of mobile devices, which may provide unnecessary rule placement. Thus, unnecessary flow rule installation causes table miss entries because the controller may not install the required corresponding flow rule entry. Accordingly, table misses cause flow setup requests, resulting in controller overhead. However, as the network grows with the high application demand of mobile devices, more flow setup requests are sent to the controller for necessary action. In response, the controller must install the corresponding flow rules to match the incoming flows. This process may consume the controller's CPU and memory, resulting in controller overhead and affecting overall network performance. Despite the growing attention to mobility-aware SDN solutions, existing studies largely overlook the controller overhead caused by rule placement under inaccurate mobility prediction. Prediction errors trigger frequent reactive flow setup requests, which significantly increase controller overhead. This work therefore focuses on mobility-aware prediction and the quantitative analysis of controller overhead, specifically examining flow setup requests generated by mobile IoT devices. Thus, the contribution of this paper is outlined below:

I. Design and implementation of a mobility-aware predictive model for SDN-enabled IoT networks.
II. Enhanced mobility prediction by integrating key decision parameters, including user preferences, bandwidth, and signal strength.
III. In-depth analysis of controller overhead under varying numbers of mobile IoT devices, capturing the impact of mobility-induced flow setup requests.
IV. Improved prediction accuracy that significantly reduces unnecessary flow rule installations compared to existing studies.
V. Complexity, sensitivity, and quantitative analysis of controller overhead, explicitly linking prediction error to reactive flow setup requests.

Related works

Accordingly, to address the installation of flow rules, various machine learning methods were conducted. A Mobi-Flow scheme for reducing rule placement is presented.[3] The scheme adopted a path estimator for predicting user locations, and a flow-manager for determining access points (APs). The scheme minimizes delay, control overhead, energy consumption, and cost. However, APs trigger additional packet-in messages due to incorrect location prediction, which increases control overhead, as noted by the authors. This limitation is also peculiar in their early version.[9,10] Also, the scheme overlooked flow setup hit ratio (FSHR2), which is critical for latency-sensitive services.[11] In addition, it is admitted that redundant flows due to wrong location predictions of APs usually increase overhead.[12] Another work presented the Health-Flow scheme, which is a criticality-aware traffic forwarding for mobile devices.[4] Machine learning is applied to determine critical flows and the location of the mobile devices. Integer Linear Programming is used for optimization by optimally selecting APs that minimize overhead. Flow-rules are placed or removed dynamically at the edge APs while considering the critical levels. This process minimizes latency, overhead, and energy consumption, and enhances network performance, but it requires accurate criticality prediction. However, it is difficult to decide because the scheme cannot provide edge nodes for multiple disease identification.[13] This indicates poor prediction of mobile devices. Also, it

considered only the healthcare system, which cannot be generalized for other systems until similar research is conducted. In contrast, to address load management in the control plane, a prediction-based control plane load reduction (CORE) is proposed.[5] Mobility prediction, rule caching, and optimized controller assignment are combined to reduce the control load. Also, CORE used a Markov Predictor for mobility prediction, a Rule-Caching Algorithm for rule placement, and a Master Controller Assignment Algorithm for timeout allocation. Compared to the existing methods, the result improved prediction accuracy and reduced control plane load, but suffered from poor prediction of mobile nodes. This is because the Markov Predictor overlooked important parameters including bandwidth, user preference, and signal strength, which reduces the prediction accuracy. Additionally, it overlooked the flow setup hit ratio (FSHR2), which is critical for latency-sensitive services.[11] This indicates that CORE needs improvement because the control load is always associated with flow setup requests, which are more frequent among large mobile users. Also, it may not scale with large networks due to high computational complexity. Likewise, a mobility-aware adaptive flow entry placement approach to address the flow-rule entry placement problem while considering the mobility of mobile nodes is presented.[6] The approach uses the Q-learning algorithm to predict mobile nodes' device locations and the AdaBoost algorithm to select heavy and active flows. The algorithm reduces table miss entries and resource expenditure but experiences poor mobility prediction of mobile devices. Also, poor prediction may cause unnecessary rule placement.[6] In addition, wrong predictions may be attributed to any machine learning-based approach if data is scarce or is plagued with variability and uncertainty; imprecision in Quality of Experience (QoE) measurement results can occur.[14] Thus, efficient ML model training is highly required.

Furthermore, machine learning approaches were applied to address the rule placement problem. DeepMonitor framework for traffic monitoring in SDN-enabled IoT is proposed.[7] The framework is developed based on a federated deep reinforcement learning approach to maximize detailed analysis of different traffic flow statistics, and Double Deep Q-network (DDQN) is employed to enable the DeepMonitor agent to reach an optimal policy rapidly. Although the framework reduces flow-table overflows, the DeepMonitor agent may not be able to learn all traffic generated by various IoT devices at the edge node, and sensitive data are at risk when shared among DDQN agents. Also, the study did not consider the mobility of mobile nodes, which generate many traffic flows. Thus, learning the traffic generated is highly required for the prediction accuracy of IoT devices. It is noted that a system should provide a high degree of prediction accuracy because an inaccurate prediction would hamper the user's experience.[15] In addition, there is a communication cost due to sharing data sets among training agents. Similarly, DeepPlace is proposed for maximizing the number of match-fields in the flow-table entry, and Markov Decision Process (MDP) is applied for managing the dynamic nature of IoT traffic flows.[8] However, the adopted algorithm may generate overhead due to computational requirements in a large-scale environment. Also, the study overlooked the mobility of mobile nodes, which generate frequent flow setup requests, thereby causing control overhead. Likewise, an eviction method using deep reinforcement learning (DRL) is proposed.[9] The method deletes unimportant rules to create space for frequent incoming rules. This process improves network performance; nevertheless, DRL may not be efficient in complex (large-scale) networks, and it overlooks the mobility of mobile devices. Also, the result was not compared with others for effective evaluation. Likewise, to minimize flow table overflows, a DRL framework for evicting Aggregate Flow Entries (AFE) is proposed.[15] The framework determines the required number of flows that can entertain with other parameters for evicting AFE. The findings reduce overflows, flow reinstallation, and control signalling overhead by 37%, 87%, and 45%, respectively. However, the framework fails to consider the mobility of mobile devices, which generates many traffic flows. Overlooking this can cause frequent flow setup requests, resulting in controller overhead. In another study, an Agg-ExTable approach is introduced to efficiently manage the usage of multiple flow tables (MFT).[16] However, despite its effectiveness, the method resulted in significant flow table space consumption, indicating a need for further optimization to reduce table usage. Additionally, Load Balancing for IoT aimed at achieving a high-performance and more reliable system was developed in,[17] while the study integrated Wireless Ad Hoc Networks to enhance Smart Cities through IoT and Cloud Computing. However, both studies failed to consider the impact of device mobility. On the other hand, to secure authentication for Internet of Medical Things (IoMT) networks, the work of[18] adopted energy-aware cluster head (CH) selection based on Artificial Democratic Cuckoo Glowworm Remora Optimization (ADCGRO) using blockchain technology. However, the study relies on a static network configuration, which may not reflect the needs of a real-time medical application. To further

address high demands on conventional communication networks due to the exponential expansion of multimedia IoT devices, the work of[19] proposed an information-centric network (ICN) based on a cooperative caching framework for content retrieval using a name instead of a location. Although the framework improved the caching process, the dynamic nature of user and essential mobility parameters was not considered. In addition, the study of[20] implemented an automated vehicle for detecting and classifying vehicles based on a chaotic equilibrium optimization algorithm with deep learning (VDTC-CEOADL) in high-resolution remote sensing images (RSIs). However, the algorithm adopted an offline process requiring substantial graphics processing unit (GPU) and computational time, specifically in large-scale networks. Collectively, the above studies overlooked device mobility, which generates frequent flow setup requests resulting in controller overhead.

Conversely, other researchers focus on mobility-aware approaches to address rule placement problems. A mobility-aware, prioritized flow-rule placement scheme is proposed.[21] The scheme proactively gave priority to flow rule placement based on mobility and flow classification while considering delay sensitivity to minimize TCAM occupancy. However, the scheme overlooked the mobility prediction of mobile devices. Although the previous works moderately improve the flow setup hit ratio (FSHR), they only maximize resource efficiency and neglect bandwidth limitations, which stop resources from being fully utilized to maximize FSHR and are critical for latency-sensitive services.[11] In another study carried out, user mobility introduces frequent flow setup latency in software-defined mobile edge networks (SDMENs), compromising latency-sensitive services.[11] Thus, the authors proposed linear programming using a polynomial-time approximation algorithm for default routing (PFS-DF) and a greedy-based heuristic algorithm for dynamic routing (PFS-DY). Although the flow setup hit ratio improves by 30.99%, the mobility prediction of mobile devices is overlooked. The influence of mobile devices generates many flow setup requests, resulting in controller overhead. Also, obtaining an optimal solution is computationally difficult due to the complexity of the problem, which relies on approximation and heuristic algorithms. Other studies focused on enhancing software vulnerability. The work of[22] presented an optimization-assisted artificial intelligence (OA-AI) model for source code vulnerability prediction, thereby helping in handling mobility of IoT devices. Furthermore, to address cyber-attacks due to the distributed and multi-user nature, the article of[23] implemented a multifactor authentication model with three stages: user authentication, biometric verification, and integrated identity, password, and biometric checks. This way, the model ensures robust confidentiality, integrity, and resistance to cyber-attacks in large-scale network settings. In addition, the study of[24] proposed an integrated strategy using deep learning techniques for anomaly detection to secure against system failure, thereby enhancing system reliability and effectiveness. However, the works of[23,24] overlooked mobility awareness in SDN-enabled IoT networks, particularly mobile devices.

In summary, the studies[7–9] overlooked the mobility of mobile devices and[11,18] ignored the mobility prediction of mobile devices. Overlooking the mobility of mobile nodes and their prediction usually generates a large number of traffic flows, and each flow requires a corresponding flow rule entry from the controller. However, as the number of mobile devices increases, packet-in messages (flow setup requests) increase. Thus, frequent flow setup requests due to the demand of mobile nodes generate communication between the switch and controller, resulting in control overhead. Likewise, the works[3–6] experienced poor prediction of mobility of mobile devices. This is attributed to overlooking important mobility history parameters, such as bandwidth allocation, user preferences, and signal strength, when predicting their location. As mobile nodes move from one location to another, their bandwidth also changes. The closer the mobile node is to the AP, the stronger the signal, and the farther from the AP, the weaker the signal. Likewise, each mobile node usually visits some locations more frequently than others. Thus, overlooking these parameters can contribute to poor prediction of mobile devices, leading to unnecessary rule placement, and the controller may not install the required flow rule to handle the incoming flows. Consequently, if flow entry does not correspond, a table miss will occur. This action will generate a flow setup request (packet-in messages), which in turn causes controller overhead. Thus, there is a need for accurate mobility prediction to enable effective rule installation, thereby minimizing table misses. Table 1 provides a summary of the existing studies while highlighting their limitations.

Materials and methods

The design and implementation incorporate a network model and begin by analyzing the performance of the SDN controller based on different numbers of mobile devices. To analyze mobility prediction and controller overhead, the methodology considered mobility prediction, flow setup requests, network state,

Table 1. Comparison of mobility prediction with existing solutions to enhance mobile device prediction.

			Parameters								
Reference	Problem address	Prediction Approach	Mobility	Location	Duration	Arrival time	Transition probability	Bandwidth	User preference	Signal strength	Weakness
[3]	Rule placement	Order-k Markov predictor and MILP	√	√	√	√	√	X	X	X	Additional packet-in messages due to incorrect location prediction, which increases control overhead
[4]	Rule placement	ML-based approach	√	√	√	√	√	X	X	X	Is not capable of providing mobile nodes for multiple disease identification due to inaccurate prediction
[5]	High control plane load in SDN	Order-m Markov predictor	√	√	√	√	√	X	X	X	It overlooked the flow setup hit ratio, resulting in high computational complexity.
[6]	Flow rule placement	Q-learning	√	√	√	√	√	X	X	X	It requires efficient ML model training
Proposed work	Controller overhead	Order-m Markov predictor	√	√	√	√	√	√	√	√	

and control message overhead. In the subsequent sections, subsection A presents the network model, and subsection B presents and derives some formulae to analyze the impact of mobility prediction, flow setup requests, network state management, and control message overhead while considering CPU and memory usage of the SDN controller performance. Finally, results were discussed based on experiments conducted to verify the proposed formulae.

Network model

Considering a network G = (C, S, A, M), where C denotes the SDN controller, S is the set of switches S = {S1, S2. . .SN}, A set of access points (APs) AP = {AP1, AP2. . .APn}, and M is the set of mobile devices (mobile stations) M = {M1, M2 . . .Mn}. Thus, each switch Si is connected to the controller, and each AP_i is connected to a switch Si. Each AP sends the received traffic from one mobile station to another via a single path. Also, each mobile node Mi is connected to a particular APi. The Mobile nodes generate traffic flow and transmit it to the network via the AP, and the AP forwards it to the switch. The controller is responsible for installing the corresponding flow rule for handling incoming flows into the flow-table of a switch. Due to the mobile nature of wireless devices, let Mi denote the probability that mobile Station (MSi) is associated with APi. Fig. 1 depicts the network topology.

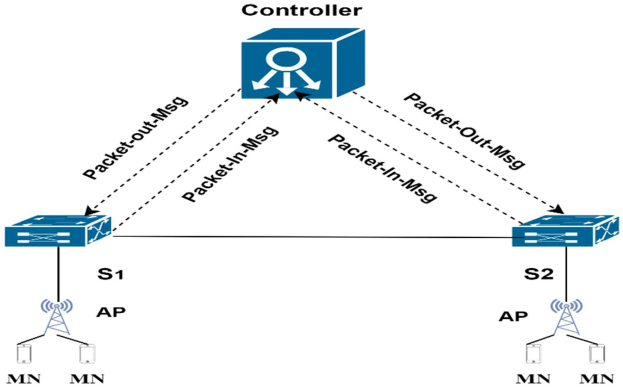

Fig. 1. Network topology.

Analysis of SDN controller overhead

The analysis is carried out based on four (Eq. (4)) performance analyses: flow setup request, network state, prediction accuracy, and controller message overhead. The research focuses on four because of the mobile nature of mobile devices. Mobile devices

move from time t_1 to t_2 across different locations within an access point (AP) coverage area and outside it while generating traffic flows. Accordingly, the controller is required to monitor their location, connection, and disconnection to update the state of the network. The update process involves frequent flow setup requests and such event-triggered control messages between the switch and the controller, resulting in controller overhead. These reasons motivated the choice of four performance metrics. The next sections explain the selected metrics:

Mobility prediction

To obtain real-time prediction of mobile devices, the MP constantly monitors flows by collecting and processing data accordingly. Network-based location information is used to determine the current location of a mobile device. The timestamp specifies when a mobile device arrives at a particular location. The duration of stay determines the time a mobile device stays at a location. The mobile device movement patterns are determined by velocity, direction, and other relevant factors (such as bandwidth, user preferences, and signal strength). Collected data from various network devices are aggregated and processed accordingly. Relevant features such as speed, direction, location, and duration of stay of a mobile device are extracted using a Markov Predictor. IoT edge devices are utilized for collecting and preprocessing data before forwarding it to the controller, and the controller processes and updates the Markov model. Real-time communication ensures low-latency communication between the IoT network devices, thereby improving flow-table utilization and minimizing controller overhead. The Markov Predictor continuously learns with the latest data and regularly updates to refine predictions.

The controller is responsible for maintaining and managing the entire network by constantly updating the network state. Thus, the load on the controller plane is determined based on the number of packet-in and packet-out messages handled by the controller during a time slot. The number of messages depends on the number of devices associated with each switch at a time slot, particularly with mobile devices. Thus, predicting the mobility of mobile devices can assist the controller in updating the network state, minimizing controller overhead. To predict the device-to-switch association, the mobility history of each mobile device is stored as prediction data. The prediction data is used to cache flow rules of specific devices in a switch and determine the optimal controller-to-switch association. Thus, to predict the future locations of mobile devices, a Markov Predictor is adopted as in.[25] Markov Predictor is the most widely used tool for predicting the future location of a mobile device based on mobility history. An order-m (*O(m)* Markov Predictor considers *m* the most recent locations of mobile devices and predicts the next location. It requires little storage and performs best among predictors for small values of *m*. For this reason, the Markov Predictor is used to predict the future locations of mobile devices and is adopted in this study.

Accordingly, to predict the location of a device $d_k(t)$, the Markov Predictor considers the following variables: $\eta_k(t) = \{\{L_{t,k}T_{t,k}\bigvee_{t,k}\}, P_{t,k}\}$ denote the mobility history of $d_k(t)$ at time-slot t, and is used to determine the mobility of a device from one access point to another to facilitate seamless handover and dynamically allocate resources for maintaining QoS while in transit. $L_{t,k} = \{l_{tk1}l_{tk2}\ldots l_{tkn}\}$ represent the locations visited by the devices and the current location, which provides fundamental data for predicting future mobility to ensure constant connectivity. $T_{tk} = \{\tau_{tk1}\tau_{tk2}\ldots\tau_{tkn}\}$ denotes the set of arrival times at the locations in L_{tk}, and L_{tk} is used to determine the inter-arrival time of each device for easy computation. $V_{tk} = \{\nu_{tkt}\nu_{tk2}\ldots\nu_{tkn}\}$ represent durations set at each location in $L_{t,k}$, and is used to determine how long the device stays in a particular location. $P_{tkij} \in P_{t,k}$ denotes the transition probability from location l_tki to location l_{tkj}, $i \neq j$, and is used for optimizing signal strength, handover management, and estimating and managing latency for real-time applications, which can be achieved by predicting the distance of the mobile devices. The output $l_{t+1,k} \in L_{t,k}$ denotes the predicted location of devices $d_k(t)$ in time slot $t+1$. The context is formulated as in Eq. (1)

$$h = L_{t,k} = (n-m+1, n) = \left\{l_{tk(n-m+1)}, l_{tk(n-m+2)}\ldots l_{tk(n-1)}, l_{tkn}\right\} \quad (1)$$

Where h denotes the location of mobile devices. The Markov Predictor extracts the context h from the input set $H_k(t)$ and examines the duration of stay V_l at location l that follows h. Therefore, the duration of stay is formulated as in Eq. (2)

$$V_l = \{\nu_{tki} | \nu_{tki} = \tau_{tki(i+1)} - \tau_{tki}, \quad where\ L_{t,k}(i-m+1, i+1) = h_l\} \quad (2)$$

Where V_l denotes the duration of stay of a mobile device at a particular location. For each V_l, when a device is shifted to a location l within Δ_t and exceeds the elapsed time τ, the conditional probability is computed as $P_t(\tau \leq \upsilon\langle\tau + \Delta\tau | h, \tau)$. Thus, Δ_t is considered the remaining time of the current time slot.

In previous studies, mobile devices are poorly predicted, resulting in unnecessary rule placement. Because important parameters such as bandwidth, user preference, and signal strength were not considered. Overlooking these parameters can lead to poor prediction of mobile devices. As a result, the controller cannot install the required flow rule to handle incoming flows, causing a table miss, and this condition generates controller overhead. To enhance prediction accuracy, this study adopted additional parameters: bandwidth (B_l), user preference (P_u), and signal strength (S_l). Accordingly, to calculate the conditional probability ($P(l|h,\tau)$ with additional parameters for any given context h and elapsed τ, the probability for a device that will move to a possible location l within $\Delta\tau$ time is given as in Eq. (3)

$$P\left(l|h,\tau\right) = P\left(l\right)P_l\ \left(\tau \le \upsilon < \tau + \Delta\tau\ |h,\tau\right).f\left(B_l,P_u,S_l\right) \quad (3)$$

Where $P(l|h,\tau)$ represent the probability change of any possible location l, $P(l)$ is the transition probability from the current context h to location l, B_l is the available bandwidth at location l, P_u is the user preference for location l, S_l is the signal strength atlocation l, and $f(B_l, P_u, S_l)$ is a function that adjusts the transition probability based on the bandwidth, user preference, and signal strength accordingly.

To predict the next location of mobile devices, the expression is formulated as in Eq. (4)

$$l_{t+1,k} = argmax_{l\in L_{t,k}}P\left(l_{t+1,k}=l\right) \quad (4)$$

Where $l_{t+1,k}$ represents the predicted location of device k at the next time slot $t+1$. The subscript $t+1$ denotes the future time slot, and k indicates the specific device. The argmax is a function that stands for "argument of the maximum", and returns the highest probability location of the variable l. Furthermore, the additional parameters allow the selection of the nearest AP that covers the predicted location with sufficient bandwidth allocation, matches user preference, and has strong signal strength. This selection is based on maximizing a score S_{AP} and is formulated as in Eq. (5)

$$S_{AP} = argmax_{AP_i\in AP_s\ coveringl_{t+1,k}}\ \left(\alpha.B_{AP_i}+\beta.P_u+\gamma.S_{AP_i}\right) \quad (5)$$

Where S_{AP} denotes the score or value assigned to a specific Access Point (AP). The term $\alpha\cdot B_{APi}$ represents the contribution of the available bandwidth at APi (B_{APi}) to the overall score, and α determines the weight of bandwidth. The term $\beta\cdot P_u$ represents the contribution of user preference (P_u) to the overall score, and β is the weight of the user preference in selecting the AP. The term $\gamma\cdot S_{APi}$ represents the contribution of the signal strength at AP_i (S_{Api}) to the overall score, and γ is the weight of the signal strength in the selection process. Therefore, this research assigned the weight values as:

$$\gamma = 0.5, \alpha = 0.3, \beta = 0.2, \text{ given } \beta+\alpha+\gamma = 1.$$

These weights were empirically adjusted based on Quality of Service (QoS) requirements and mobility objectives. The signal strength ($\gamma = 0.5$) is prioritized for link reliability, bandwidth ($\alpha = 0.3$) for traffic demand to support QoS, and user preference ($\beta = 0.2$) for policy compliance. This provides normalization in the network. Finally, to update the location of the device-switch association and disassociation, the $Z_{ik}(t)$ can be set to 1 if switch Si is associated with the selected AP. Thus, mathematically,the expression is formulated as in Eq. (6)

$$Z_{ik}\ (t) = 1 \quad (6)$$

Where the term $Z_{ik}\ (t)$ represents the binary variable indicating the association between mobile device k and switch s_i at time t. Thus, if $Z_{ik}\ (t) = 1$, it means that device k is associated with switch s_i at time t and if $Z_{ik}\ (t) = 0$, it means that device k is not associated with switch s_i at time t.

The mobility prediction receives inputs from the proposed MAP algorithm. It is followed by extracting the location of a mobile device from the mobility history at a particular time slot t, and computes the possible location of the mobile device in lines 2-6 of the MPA algorithm. The conditional probability of a given mobile device with additional parameters at a location, starting from l with the current state h at time t, is determined in lines 7 -14 of MPA. Similarly, the next location of the mobile device is predicted based on computed probabilities in lines 16-18 of the MPA. Likewise, the nearest access point (AP) that covers the predicted location of the mobile device and matches the radio access capability of $d_k(t)$ is selected as indicated in line 19. Finally, to update device location, suppose the mobile device is associated with the selected AP, the algorithm sets $Z_{ik}(t) \leftarrow 1$, but if the mobile is not associated with the AP, $Z_{ik}(t) \leftarrow 0$. The Predictor backtracks to $O(m-1)$ and determines the most frequent location based on the history as shown in lines 21-25. Therefore, the algorithm executes successful future location prediction and association of mobile devices with suitable APs and the network is updated accordingly.

Algorithm 1: Mobility Prediction Algorithm (MPA).

Inputs: $\{H_k(t-1)\}_{k=1}^{n}, h, \Delta\tau$
Output: $\{z_k(t)\}_{k=1}^{n}$
Procedure:
Get the required inputs
For each device d_k, $k = 1$to n:
  #Extract location history
  Get $L_{t-1,k}$ from $H_k(t-1)$
  #Compute Duration of Stay
  V_l at possible location l using Eq. 2
  For each location $l = 1$to m:
    If $d_k(t)$ is shifted to location l within Δτ AND t > Δτ **Then**
      Compute $P(l|h,\tau)$ using Eq. 3
      Shift l within Δτ based on elapsed time τ
      Get Δτ at time-slot τ, Get h and elapsed τ
      Move location l within Δτ
    End If
  End For
  #Predicting next location of mobile node
  Compute $l_{t+1,k}$ using Eq. 4
  # Selecting the best AP
  Compute S_{AP} using Eq. 5
  Select AP based on prediction matching radio access of $d_k(t)$
  #Updating device-AP association
  If d_k is associated with AP, **then**
    $Z_{ik}(t) = 1$ and update the status of $Z_{ik}(t)$
  Else
    $Z_{ik}(t) = 0$ and backtracks to *O(m-1)*
  End If
End For
End of Algorithm

Flow setup request

Flow setup request is a packet-in message to the controller for updating the network state triggered by a mobile node. It may occur as a result of proactive rule installation, reactive rules installation (table miss) or flow-table overflow. In Fig. 1, at time t_1, a mobile node (MN) may send traffic flow (f) to MN2 or MN3 that are connected to the same AP or to a different AP. Accordingly, the controller has to update and manage their communication since the MNs are in different locations, either in the same AP or other AP coverage. Let T_{DI} denote the data flow initiation time. As the controller receives a request, it processes it and forwards it to the MN. Let T_{CP} denote controller processing time, and T_{PC} denotes path calculation time. It is assumed that selected paths can meet the flow QoS demand. Thus, the controller selects a route based on flow QoS. Also, let TCMP denote control message propagation time, which is the time the controller sends a message to a relevant switch and attached AP along the route. Eq. (7) demonstrates how to obtain flow setup requests.

$$F_{ST} = \sum T_{DI} + T_{CP} + T_{PC} + T_{CMP} + T_{FE} \quad (7)$$

Where F_{ST} denotes the total flow setup time, $\sum$ represents the total summation for the whole flow setup requests, T_{DI} indicates data ingress time, which is the time taken for a packet to arrive at the switch. T_{CP} serves as controller processing time, which is the time taken to process a packet-in message by the controller. T_{PC} represents the propagation and communication time between the switch and the controller. T_{CMP} denotes computation and matching processing time, which is the time taken to calculate a forwarding decision that matches flow rule entries, while T_{FE} indicates flow entry installation time, which is the time taken to install the flow rule into the flow-table of a switch. Therefore, this equation determines the total summation of flow setup time in an SDN-enabled IoT environment based on different mobility scenarios. This allows the analysis of the controller overhead by capturing incoming packets, controller processing, communication between the switch and controller, mobility-aware decision-making, and flow rule installation.

Network state management

The Network State Control Management (NSM) is responsible for managing and updating the network state while monitoring switches, AP, MN location, association, and disassociation of MN with associated AP. Let S_{ti} denote the network state at time t_i and each mobile node is associated with information including location, connection status, and associated AP. Let D(ti) denote the information associated with mobile device 'i' at time 't'. However, MN may change its location at any time due to mobility behavior. Thus, let represent the movement of mobile devices 'i' as a function Mi (ti), showing the device's location at time 't_i'. Therefore, the relationship between the mobile devices and the access points can be represented by function A_i (t_i), indicating the access point is connected to a particular mobile device i at time t_i. Whenever the status of a device changes, an update takes place accordingly. Let $U_i(t_i)$ denote the update process for mobile device i at time ti. Thus, the calculate the total consumption by the NSM can be formulated as in Eq. (8)

$$S(t) = \{D_i(t), A_i(t) \forall Device_i\} \quad (8)$$

Where *S(t)* denotes the network state at time *t*, which is the mobility-aware status for the mobile nodes, and *t* indicates the time at which the network condition is updated. $D_i(t)$ represents the mobility-associated state of device *i* at time *t* of the predicted next location, and *i* denotes an index of a specific mobile node. $A_i(t)$ denotes the association state of device *i* at time *t*, which indicates its current connection with AP. ∀ indicates the present state of IoT devices in the network. This way, Eq. (8) will capture the mobility status of all associated IoT devices at any given

time. This enables mobility prediction, proactive rule installation, and controller overhead minimization.

The network state changes as the mobile devices move from one location to another, which changes the status of the mobile devices as well. Thus, for each change of mobile device '*i*', an update process is triggered, as in Eq. (9)

$$D_i\,(t+\nabla t) = u_i\,(t) \tag{9}$$

Where $D_i(t+\nabla t)$ represent the predicted mobility status of device i at the future time. ∇t denotes the prediction increment time. $u_i(t)$ denotes the mobility predicted status of device i at time t. This enables Eq. (9) to predict future mobile node status based on data history. This way, the controller can anticipate device mobility and pre-install flow rules, which reduces controller overhead. Usually, the NSM is complex because several mobile devices interact by generating a lot of traffic flows. This condition triggers updates due to mobility, connection, change of location, bandwidth, signal, and other network events.

Control message overhead

Control Message Overhead (CMO) refers to an additional processing load incurred on the controller as a result of managing SDN-enabled IoT, with mobility association and disassociation of mobile nodes.[1] In[15] Q-learning algorithm experienced poor prediction due to omission of important parameters such as bandwidth allocation, user preference, and signal strength. This may cause poor predictions, leading to unnecessary rule placement and enabling flow setup overhead.[14] Wrong predictions may be attributed to any machine learning-based approach, particularly if data is scarce or overwhelmed with variability and uncertainty, causing poor Quality of Experience (QoE) measurement. Thus, the controller may install an incorrect flow rule when a mobile node is wrongly predicted, which may lead to an unnecessary flow rule placement, resulting in table misses and flow-table overflow. Consequently, frequent flow setup requests would be triggered, causing overhead for the controller. Thus, the element of CMO can be stated as $C_{I,}$ denoting the message used in conveying information about the network condition, like association and disassociation of mobile nodes and other events. Also, let 'E' denote the set of all potential mobile nodes, and 'T_{CMO}' denote the total control message overhead. Accordingly, the T_{CMO} can be formulated as in Eq. (10)

$$T_{CMO} = \sum (C_i) \forall \textit{ event } i \textit{ in } E \tag{10}$$

Where T_{CMO} denotes total control messages, which serve as the total computation cost carried out by the controller. $\sum$ represents the overall summation of the control-related messages; Ci provides the cost of control messages for event i (packet-in and packet-out messages); and i represents the control event for an individual mobile node. E represents control events such as mobile node association, disassociation, and flow setup requests, while ∀ denotes summation over every event within the set E. This equation enables understanding of how mobility generates flow setup requests that contribute to controller overhead, which can assist in reducing controller overhead. Naturally, the T_{CMO} can augment over time due to mobile nodes' mobility. Thus, as the mobile nodes increase, the TCMO increases, resulting in controller overhead. Accordingly, the relationship between the previous and the current network state can be complex due to several factors, including the efficiency of control messages, the strategy of managing the network state, and prediction accuracy, which is critical.

Quantitative analysis. To compute a quantitative analysis of the controller, let M(t) be the number of mobile devices moving at time t, PER denotes prediction error rate, $F_e(t)$ represent the expected number of flow setup requests; F_{corr} represents the number of requests for correct prediction; and F_{err} number of requests for wrong prediction. In SDN-enabled IoT, mobile devices (M) are expected to move across different coverage areas of access points (APs) at time t. Therefore, the controller overhead is driven by the mobility-induced flow setup requests. When M moves from one AP to another, correct mobility prediction at time t allows proactive rule installation, which mitigates table-miss and reduces controller interaction. On the other hand, prediction error causes unnecessary rule placement, thereby triggering a reactive flow setup request to the controller, resulting in controller overhead. Thus, $F_e(t) = M(t)((1-PER)F_{corr} + PERF_{err})$, where wrong prediction ($F_{err} > F_{corr}$). Therefore, the controller message overhead (T_{CMO}) can be expressed as $T_{CMO} = m_c \cdot F_e(t)$ where m_c denotes the average number of control messages generated per flow setup request. Consequently, the additional controller overhead due to prediction errors can be expressed as $\Delta F_e(t) = m_c \cdot \text{PER} \cdot M(t)(F_{err} - F_{corr})$, demonstrating a linear relationship between prediction error and controller overhead. Consequently, this effect can be intensified when the network experiences high mobility density. This highlights that accurate prediction directly minimizes controller overhead.

Complexity analysis. To analyse the computational complexity, let n denote the number of mobile devices ($k = 1, \ldots, n$) and $m$denote the number of possible user locations per device ($l = 1, \ldots, m$). The Eqs. (2) and (4) of the proposed MPA run in constant time $O(1)$, as do the association update and backtracking decisions. In Line 2, the outer loop iterates over all devices, executing n times. For a fixed device d_k, lines 3–6 compute the duration of stay from the location history in $O(1)$. In Line 7, the inner loop iterates over all candidate locations $l = 1 \ldots m$, executing m times. Lines 8–13 compute conditional probabilities and update the user's location, each in $O(1)$. Hence, the iteration cost for device $d_k$is $O(m)$. Lines 15–19 predict the next location Eq. (4) and select the best AP Eq. (5) in $O(1)$, while lines 20–25 update the device–AP association and handle possible backtracking, also in $O(1)$. Therefore, the total cost for device $d_k$is: $O(1) + O(m) + O(1) + O(1) = O(m)$, where $O(m)$denotes linear growth with respect to the number of candidate locations. To process all devices, the total cost becomes: $T(n, m) = n \times O(m) = O(nm)$. Thus, the overall time complexity of the MPA is: $O(nm)$, which denotes linear growth with respect to both the number of mobile devices and the number of possible user locations. This indicates that the MPA scans and processes each device across all candidate locations once, demonstrating that the complexity scales linearly in practical SDN-enabled IoT mobility scenarios.

Sensitivity analysis. To carry out sensitivity analysis, the following parameters were considered: prediction interval ($\Delta\tau$), number of mobile devices (n), number of user locations (m), history of mobility duration (h), and prediction error rate (PER) sensitivity. When a mobile device moves to a new location area of AP coverage at time t, the controller must update the current location of the mobile device. Thus, if m is wrongly predicted, the flow rule would be installed reactively by the controller. This increases controller overhead. Firstly, small $\Delta\tau$ executes frequent prediction updates, frequent controller interactions, and good responsiveness to mobility. Conversely, large $\Delta\tau$ minimizes computation overhead, but increases the risk of inaccurate predictions. Therefore, moderate $\Delta\tau$ can provide balanced execution between prediction accuracy and controller overhead. Secondly, as the number n increases, the prediction cost and the controller overhead increase linearly due to frequent flow setup requests. Thirdly, large m increases the prediction search space per device and is restricted within the AP range. Conversely, small m is moderate within the coverage area of the AP. Fourthly, small h executes fast computation with less reliability, while large h enhances prediction with a slight increase in computation and memory utilization. Finally, higher *PER* executes frequent reactive flow setup requests with increased controller overhead. On the other hand, lower *PER* executes proactive rule installation with reduced flow setup requests. Accordingly, the sensitivity analysis demonstrates that the proposed MPA is linear with m and n, whereas its performance remains sensitive to the prediction interval and prediction accuracy. The moderate values of $\Delta\tau$ and h attain a balance between accuracy and controller overhead, while higher *PER* substantially increases controller overhead due to frequent reactive flow setup requests.

Simulation setup

The study is conducted with a Mininet Wi-Fi simulated environmental setup using an Intel (R) Core (TM) i7-1255U, 1.70 GHz, and 8.00GB memory. For testing and evaluation, the Ryu open-source controller is adopted for creating real-time network topology scenarios. The network topology was built using Python code to evaluate the experiment, and OpenFlow switches were adopted to connect different access points (APs). Similarly, mobile nodes are connected to APs via wireless equipment on Mininet Wi-Fi. Mobile nodes are predicted from different locations based on their parameters while generating several traffic flows to APs using the Distributed Internet Traffic Generator (DITG).

Results and discussion

Hewlett Packard Enterprise,[26] recommends supporting approximately 30–50 devices per AP for real-world deployments to maintain network stability and quality of service. Thus, this study adopts the configuration of 25 mobile devices per AP, enabling realistic modeling for practical deployment scenarios. The experiment revealed that CPU and memory usage increase as the number of mobile nodes and active flow entries rises. Table 2 presents five different scenarios, each based on the controller's CPU and memory usage under varying behaviors of mobile devices. In each scenario, the experiment examines different sets of mobile devices along with their total active flow entries. The prediction accuracy was observed to be 100% across all the scenarios, demonstrating the effectiveness of the Predictor model as mobile nodes changed locations within the network. Fig. 2 illustrates prediction accuracy for different sets of mobile nodes; Fig. 3 shows prediction accuracy under varying flow entries; and Table 2 shows the controller's CPU and memory usage for varying numbers of mobile nodes.

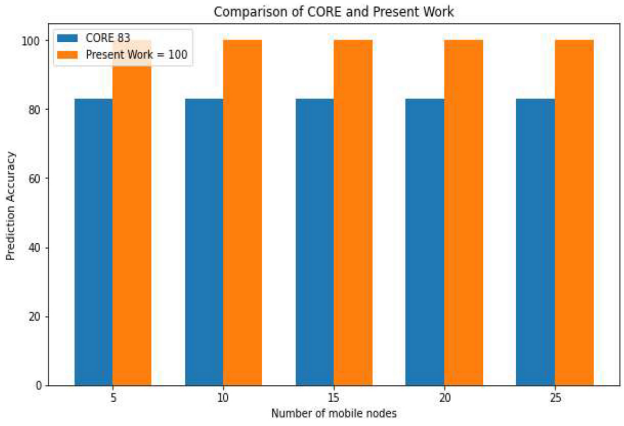

Fig. 2. Prediction accuracy with different mobile nodes.

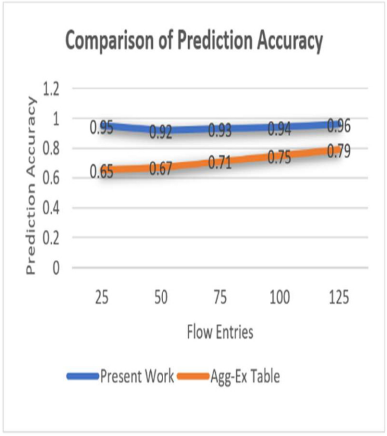

Fig. 3. Prediction accuracy across varying flow entries.

The graph in Fig. 2, compares the prediction accuracy of CORE and the Present Work across different numbers of mobile nodes (5, 10, 15, 20, 25). The x-axis represents the number of mobile nodes, while the y-axis indicates the prediction accuracy, ranging from 0% to 100%. It appears the "Present Work" outperforms the "CORE" significantly, achieving an accurate prediction accuracy of 100% across all scenarios tested, regardless of the number of mobile nodes. In contrast, the "CORE" shows slightly lower prediction accuracy across different numbers of mobile nodes, with 83.72%. Thus, Fig. 2 highlights the effectiveness of the Present Work, which consistently predicts mobile devices with 100% accuracy irrespective of their number, demonstrating its robustness compared to the "CORE" model, which has a lower and slightly variable prediction accuracy.

In Fig. 3, a comparison of prediction accuracy across varying flow entries between the Present Work and the Agg-ExTable is presented. At 25 flow entries, the Present Work achieves an accuracy of 0.95, while the Agg-Ex-Table records 0.65. The Present Work consistently demonstrates higher and more stable prediction accuracy, ranging from 0.92 to 0.96, whereas the Agg-Ex Table shows a slower improvement, with accuracy values ranging from 0.65 to 0.79. These results indicate that the Present Work outperforms the Agg-Ex Table in all scenarios, highlighting its robustness and scalability in managing flow entries. In contrast, the Agg-Ex Table exhibits lower accuracy and slower adaptation, particularly under increasing flow loads. The consistent performance of the Present Work underscores its efficiency and reliability in handling flow entries within dynamic network environments.

In the first scenario, the study periodically monitors the controller's CPU and memory usage for 5 seconds. Fig. 4 of scenario1 shows the CPU and memory usage with 5 mobile nodes, corresponding to 1467 active flow entries. The highest CPU usage observed at the initial time is 1.3%, while the memory usage remains constant.

at 33.5% throughout the periodic time of 5 seconds. Also, it was observed that the CPU and memory usage are relatively low because the number of mobile nodes is small, resulting in a small number of flow setup requests to the controller. In the third scenario, Fig. 5 of scenario 2, shows 10 mobile nodes, corresponding to 1888 active flow entries. The CPU usage is 2.1% at the initial time, while the memory usage is 35.2% and remains the same. It is also noted that as the number of mobile nodes increases from 5 to 10, active flow entries increase, resulting in higher CPU and memory usage compared to Fig. 4 of scenario 1. Fig. 6 of the third scenario presents the performance metrics with 15 mobile nodes, corresponding to 3205 active flow entries. It is observed

Table 2. Experimental scenario.

Scenario	Total Active Flows Entry	Mobile nodes	Periods	Prediction Accuracy	CPU Usage (%)	Memory Usage (%)	Flow table utilization
1	1467	5	5	100	1.3	33.5	85
2	1888	10	10	100	2.1	35.2	85
3	3205	15	10	100	4.4	36	100
4	7189	20	10	100	5.5	35.1	300
5	8331	25	10	100	38.8	41.5	400

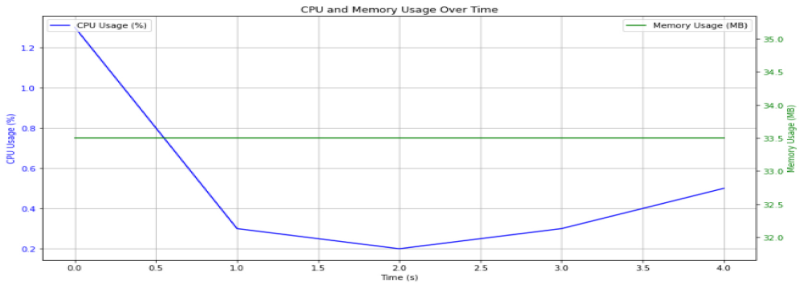

Fig. 4. 1467 Active flow entries with 5 mobile nodes.

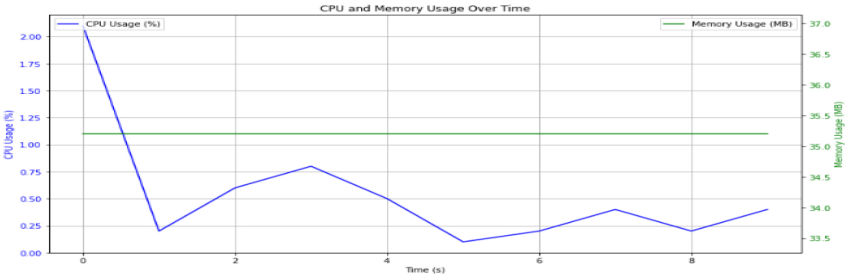

Fig. 5. 1888 Active flow entries with ten10 mobile nodes.

that active flow entries increase significantly because mobile nodes increase from 10 to 15. The CPU and memory usage increased substantially to 4.4% and 36%, respectively, compared to Fig. 5. The substantial increase in CPU and memory usage reflects the growing demand for the network's resources. In the fourth scenario, the mobile nodes of Fig. 7 increased to 20, corresponding to 7179 active flow entries increase. CPU utilization sharply rises to 5.5%, while memory utilization slightly decreases to 0.9%, indicating the

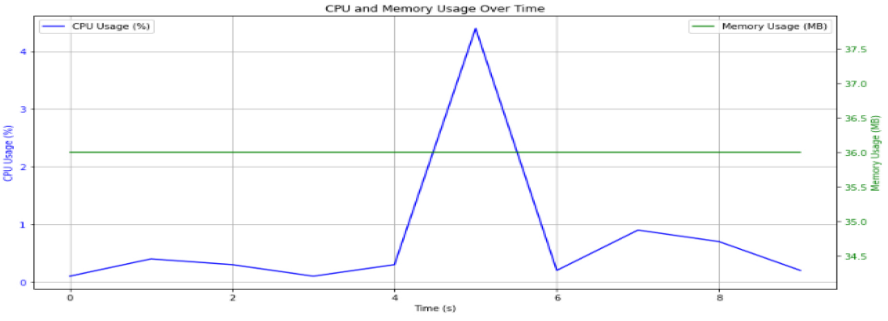

Fig. 6. 3205 Active flow entries with 15 mobile nodes.

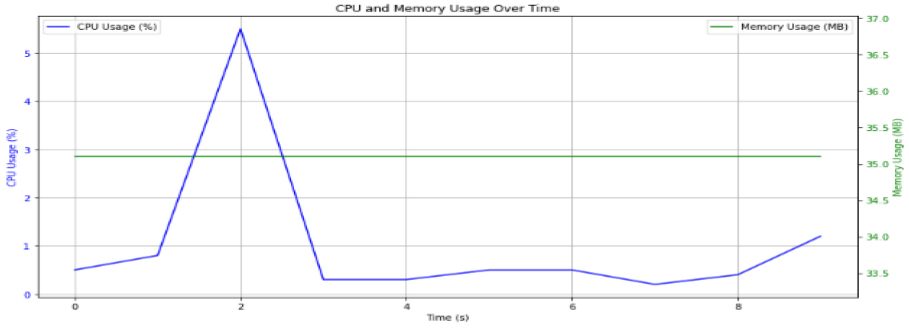

Fig. 7. 3205 Active flow entries with 15 mobile nodes.

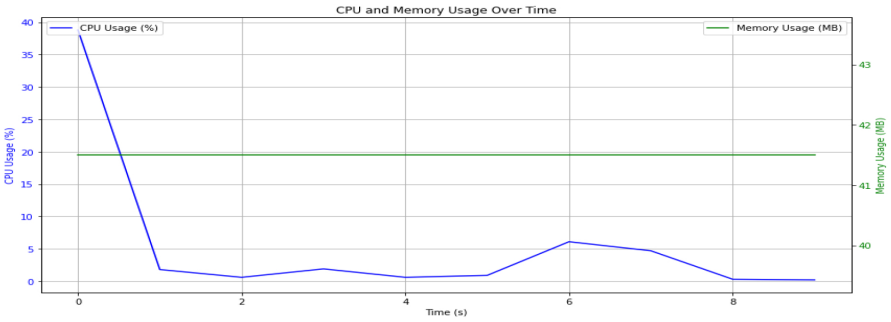

Fig. 8. 8331 Active flow entries with 25 mobile nodes.

scalability challenges as the network expands compared to Fig. 5. Finally, Fig. 8 of the fifth scenario represents 25 mobile nodes as the highest number, corresponding to 8331 active flow entries. It was observed that the CPU and memory usage are 3.8% and 41.5%, respectively. In general, resource utilization of both CPU and memory increases exponentially with rising network demand, as illustrated in Figs. Figs. 4 to 8. As the number of mobile devices grows, the volume of flow setup requests directed to the controller also increases, leading to higher controller overhead. This overhead can, in turn, degrade overall network performance by increasing latency and reducing responsiveness.

Limitation and future research

Research is a continuous process from one generation to another. The application of IoT has been increasing, particularly with the advent of mobile devices. Consequently, IoT devices generate a lot of traffic in the network under limited switch memory for flow tables and switch-controller communication. Therefore, the incoming traffic (flows of packets) needs to be managed effectively to minimize flow-table utilization, thereby reducing controller overhead. This study investigates the mobility-aware prediction to minimize controller overhead. However, the limitation of the study is as follows:

Proactive rule installation

It has been observed that in a proactive flow rule installation method, resource utilization increases as a result of pre-installing several rules. This may unnecessarily consume limited switch memory compared to the reactive method of flow rule installation. However, the reactive method reduces memory utilization compared to proactive, but is signaling intensive, which could consume SDN controller resources (e.g. CPU), particularly with mobility prediction errors. Furthermore, it increases the latency tolerance and reliability requirements of the control channel and control plane. Therefore, future research will integrate proactive and reactive methods, which can dynamically adapt to network changes, thereby minimizing flow setup requests and controller overhead.

Single controller

The study employed a single controller for easy analysis of controller overhead in an SDN-enabled IoT environment. However, it may experience bottlenecks as the network expands, particularly single point failure and high computational decision making. Thus, future research will consider multiple controllers to enhance scalability, reliability, performance, and resilience against single points of failure.

Conclusion

Software defined networking is a flow-based network that requires the controller to install the corresponding flow rule into the switch for handling incoming flows. Unfortunately, the network increases as the number of mobile devices increases, resulting in more resource utilization due to rule placement. Thus, software-defined networking (SDN-enabled IoT) is leveraged to manage IoT mobile devices due to its programmability and scalability for managing the global network. However, the SDN controller and switch can be overutilized due to flow setup requests and the limitation of switch memory. Therefore, this study analyzed how mobile nodes' flow setup requests, control messages overhead, and predictions affect SDN controller CPU and memory usage. The prediction was confirmed to be 100% compared to CORE, minimizing unnecessary flow rule installation. Similarly, the analysis revealed that CPU and memory utilization increase exponentially as the network demand increases with more mobile nodes. Thus, as the number of mobiles increases, more flow

setup requests are sent to the controller, leading to increased controller overhead. Furthermore, sensitivity analysis shows that low prediction error reduces controller overhead due to proactive rules installation, while high prediction error increases controller overhead due to frequent flow setup requests as a result of reactive controller response. Therefore, future research should consider integrating proactive and reactive methods, which can dynamically adapt to network changes, thereby minimizing frequent flow setup requests and controller overhead. In addition, multiple controllers should be explored to avoid single points of failure, making the system resilient to network changes.

Acknowledgment

The authors would like to express their gratitude to Universiti Teknologi Malaysia (UTM) for the financial support provided through the research grant PY/2024/01535, which has facilitated this study.

Authors' declaration

- Conflicts of Interest: None.
- We hereby confirm that all the Figures and Tables in the manuscript are ours. Furthermore, any Figures and images that are not ours have been included with the necessary permission for republication, which is attached to the manuscript.
- No animal studies are present in the manuscript.
- No human studies are present in the manuscript.
- Ethical Clearance: The project was approved by the local ethical committee at the University of Teknologi Malaysia.

Authors' contributions statement

K.B.A. supervised the research; B.I. provided the conceptual framework and design; M.N.Y. carried out proofreading and editing; F.B.M. conducted data analysis and interpretation; and finally, A.S. developed the manuscript and conducted the experiments.

References

1. Isyaku B, Abu Bakar K, Abdulrahman S, Yusuf MN, Muchtar FB, Ghaleb FA. Mobile Device Influence on SDN Controller Performance in IoT-Managed Software-Defined Wireless Networks. In: International Conference of Reliable Information and Communication Technology (IRICT 2023). Springer, Cham; 2024. p. 62–72. https://doi.org/10.1007/978-3-031-59707-7_6.
2. Isyaku B, Mohd Zahid MS, Bte Kamat M, Abu Bakar K, Ghaleb FA. Software defined networking flow table management of OpenFlow switches performance and security challenges: A survey. Future Internet. 2020 Aug 31;12(9):147. https://doi.org/10.3390/fi12090147.
3. Bera S, Misra S, Obaidat MS. Mobi-Flow: Mobility-Aware Adaptive Flow-Rule Placement in Software-Defined Access Network. IEEE Trans Mob Comput. 2019 Aug 1;18(8):1831–42. https://doi.org/10.1109/TMC.2018.2868932.
4. Misra S, Saha R, Ahmed N. Health-flow: criticality-aware flow control for SDN-based healthcare IoT. In: GLOBECOM 2020-2020 IEEE Global Communications Conference 2020 Dec 7 (1–6). IEEE. https://doi.org/10.1109/GLOBECOM42002.2020.9348058.
5. Maity I, Misra S, Mandal C. CORE: Prediction-Based Control Plane Load Reduction in Software-Defined IoT Networks. IEEE Trans Commun. 2021 Mar 1;69(3):1835–44. https://doi.org/10.1109/TCOMM.2020.3043760.
6. Huang G, Ullah I, Huang H, Kim KT. Predictive mobility and cost-aware flow placement in SDN-based IoT networks: a Q-learning approach. J Cloud Comput. [Internet]. 2024 Jan 25;13(1):26. https://doi.org/10.1186/s13677-024-00589-w.
7. Nguyen TG, Phan T V., Hoang DT, Nguyen TN. So-In C. Federated Deep Reinforcement Learning for Traffic Monitoring in SDN-Based IoT Networks. IEEE Trans Cogn Commun Netw. 2021 Dec 1;7(4):1048–65. https://doi.org/10.1109/TCCN.2021.3102971.
8. Nguyen TG, Phan T V., Hoang DT, Nguyen HH, Le DT. DeepPlace: Deep reinforcement learning for adaptive flow rule placement in Software-Defined IoT Networks. Comput Commun. 2022 Jan 1;181:156–63. https://doi.org/10.1016/j.comcom.2021.10.006.
9. Jiménez-Lázaro M, Berrocal J, Galán-Jiménez J. Deep reinforcement learning based method for the rule placement problem in software-defined networks. In IEEE/IFIP Network Operations and Management Symposium (NOMS) 2022 Apr 25 (1–4). IEEE. https://doi.org/10.1109/NOMS54207.2022.9789906
10. Bera S, Misra S, Obaidat MS. Mobility-Aware Flow-Table Implementation in Software-Defined IoT. In: 2016 IEEE Global Communications Conference (GLOBECOM). IEEE;2016:1–6. https://doi.org/10.1109/GLOCOM.2016.7841995.
11. Zeng Y, Ye B, Tang B, Lu S, Xu F, Guo S, *et al.* Mobility-Aware Proactive Flow Setup in Software-Defined Mobile Edge Networks. IEEETrans Commun. 2023 Mar 1;71(3):1549–63. https://doi.org/10.1109/TCOMM.2023.3238396.
12. Jurado-Lasso FF, Marchegiani L, Jurado JF, Abu-Mahfouz AM, Fafoutis X. A survey on machine learning software-defined wireless sensor networks (ML-SDWSNS): Current status and major challenges. IEEE Access. 2022 Feb 22;10:23560–92. https://doi.org/10.1109/ACCESS.2022.3153521.
13. Saha R, Ahmed N, Misra S. SDN-controller triggered dynamic decision control mechanism for healthcare IoT. In: 2021 IEEE Global Communications Conference (GLOBECOM) 2021 Dec 7 (1–6). IEEE. https://doi.org/10.1109/GLOBECOM46510.2021.9685911.
14. Mitra K, Zaslavsky A, Ahlund C. Context-Aware QoE Modelling, Measurement, and Prediction in Mobile Computing Systems. IEEE Trans Mob Comput. 2015 May 1;14(5):920–36. https://doi.org/10.1109/TMC.2013.155.
15. Zang J, Raza SM, Choo H, Byun G, Kim M. Deep reinforcement learning driven aggregate flow entries eviction in software defined networking. In 2023 International Conference on Information Networking (ICOIN) 2023 Jan 11 (282–286). IEEE. https://doi.org/10.1109/ICOIN56518.2023.10049020.

16. Wang C, Youn HY. Entry aggregation and early match using hidden Markov model of flow table in SDN. Sensors. 2019 May 21;19(10):2341. https://doi.org/10.3390/s19102341.
17. Abed MM, Younis MF. Developing load balancing for IoT-cloud computing based on advanced firefly and weighted round robin algorithms. Baghdad Sci J. 2019;16(1):130–139. http://dx.doi.org/10.21123/bsj.2019.16.1.0130.
18. Yuvarani R, Mahaveerakannan R, Thanarajan T, Kartheesan L. Energy-aware cluster head optimization and secure blockchain integration for heterogeneous 6G-enabled IoMT networks. Sci. Rep. 2025 Aug 16;15(1):30009. https://doi.org/10.1038/s41598-025-15462-2.
19. Mahaveerakannan R, Tamilvizhi T, Rayen S, Khalaf O, Hamam H. Information centric networking based cooperative caching framework for 5G communication systems. Comput. Mater. & Contin. 2024;80(3):3945. https://doi.org/10.32604/cmc.2024.051611.
20. Alotaibi Y, Nagappan K, Thanarajan T, Rajendran S. Optimal deep learning based vehicle detection and classification using chaotic equilibrium optimization algorithm in remote sensing imagery. Sci. Rep., 2025 May 23;15(1):17921. https://doi.org/10.1038/s41598-025-02491-0.
21. Kyung Y. Mobility-aware prioritized flow rule placement in software-defined access networks. In 2021 International Conference on Information Networking (ICOIN) 2021 Jan 13 (59–61). IEEE. https://doi.org/10.1109/ICOIN50884.2021.9333854.
22. Vanam RR, Yadavali VV, Elumalai S, R S. Software vulnerability detection in source code using superb fairy-wren deep transformer guided model for next generation software security. In: 2025 6th International Conference on Inventive Research in Computing Applications (ICIRCA). IEEE; 2025. p. 101–106. https://doi.org/10.1109/ICIRCA65293.2025.11089899.
23. Ramya M, Tamilvizhi T, Navaneethakrishnan SR. Enhancing Cloud Security with Multi-Factor Authentication and Optimized Encryption Algorithm. In: 2025 6th International Conference on Electronics and Sustainable Communication Systems (ICESC). Coimbatore, India: IEEE; 2025. p. 1033–1039. https://doi.org/10.1109/ICESC65114.2025.11212624.
24. Vanam RR, Krishnama CR, Elumalai S, R S. Enhancing Software Reliability through Anomaly Detection: Implementing Variational Autoencoders for Real-time Performance Monitoring and Error Prediction. In: 2025 6th International Conference for Emerging Technology (INCET). BELGAUM, India: IEEE; 2025. p. 1–8. https://doi.org/10.1109/INCET64471.2025.11140983.
25. Maity I, Dhiman R, Misra S. Mobiplace: Mobility-aware controller placement in software-defined vehicular networks. IEEETrans Veh.Technol. 2021 Jan 6;70(1):957–66. https://doi.org/10.1109/TVT.2021.3049678.
26. Hewlett Packard Enterprise (2019). How many devices can be connected to a single access point? Aruba Community. HPC; 2019.

حليل تنبؤي مدرك للحركة لعبء المتحكم في إنترنت الأشياء المُمكَّن بالشبكات المعرفة برمجيًا (SDN)

عبد الرحمن سعيدو[2،1]، كمال الرولنظام بن أبو بكر[1]، بابانغيدا إيسياكو[3]، محمد نورا يوسف[4]، فرخانة بنت مختار[1]

[1] كلية الحوسبة، الجامعة التكنولوجية الماليزية، جوهور، ماليزيا، 81310 جوهور باهرو، جوهور، ماليزيا.
[2] قسم علوم الحاسوب، كلية العلوم والتكنولوجيا، البوليتكنيك الفيدرالي في بالي، ولاية تارابا، نيجيريا.
[3] قسم علوم الحاسوب، كلية الحوسبة وتكنولوجيا المعلومات، جامعة سلي لاميدو، ص.ب 048، كافين هاوسا، ولاية جيغاوا، نيجيريا.
[4] قسم علوم الحاسوب، كلية الحاسوب وتكنولوجيا المعلومات، جامعة أبو بكر تافاوا باليوا، باوتشي، ص.ب 0284، نيجيريا.

الخلاصة

إن النمو السريع والازدياد المستمر في حركة أجهزة إنترنت الأشياء (IoT) المتنقلة يؤديان إلى زيادة كبيرة في طلبات إعداد التدفقات (Flow Setup Requests) في الشبكات المعرفة برمجيًا (SDN). إذ إن التنقل المتكرر للأجهزة المتنقلة بين نقاط الوصول يتسبب في عمليات متكررة لتثبيت قواعد التدفق في المحولات العاملة ضمن مستوى البيانات (Data Plane Switches)، مما يؤدي إلى زيادة مفرطة في العبء الواقع على المتحكم (Controller) في ظل السعة المحدودة لذاكرة جداول التدفق. كما أن عدم دقة التنبؤ بالحركة في الأساليب الحالية يزيد من تفاقم هذه المشكلة من خلال التسبب في وضع قواعد غير ضرورية وطلبات متكررة لإعداد التدفقات. تبحث هذه الدراسة في : تأثير التنبؤ المدرك للحركة على العبء الواقع على المتحكم من خلال سيناريوهات تجريبية تستخدم مجموعات مختلفة من أجهزة إنترنت الأشياء المتنقلة، مع الأخذ في الاعتبار استهلاك وحدة المعالجة المركزية (CPU) والذاكرة الخاصة بالمتحكم. وقد تم توظيف نموذج استباقي للتنبؤ بالحركة بهدف تقليل التفاعلات التفاعلية (Reactive Interactions) مع المتحكم. وأظهر نموذج التنبؤ المقترح دقة تقارب 100% مقارنة بالنموذج المرجعي، مما أسهم بشكل ملحوظ في تقليل طلبات إعداد التدفقات غير الضرورية وخفض العبء الواقع على المتحكم. علاوة على ذلك، أوضح تحليل الحساسية أن انخفاض خطأ التنبؤ يؤدي إلى تقليل الحمل على المتحكم، في حين أن ارتفاع خطأ التنبؤ يؤدي إلى زيادة العبء عليه، مما يؤكد أهمية التنبؤ الدقيق بالحركة لتحقيق قابلية التوسع في شبكات إنترنت الأشياء المعتمدة على الشبكات المعرفة برمجيًا.

الكلمات المفتاحية: المتحكم، إنترنت الأشياء، الحركة، التنبؤ، الشبكات المعرفة برمجيًا (SDN).