Received: 3 September 2021 | Revised: 18 November 2021 | Accepted: 5 January 2022

IET Communications

DOI: 10.1049/cmu2.12328

ORIGINAL RESEARCH PAPER

An intelligent method for reducing the overhead of analysing big data flows in Openflow switch

Mahdi Abbasi[1] | **Shima Maleki**[1] | **Gwanggil Jeon**[2] | **Mohammad R. Khosravi**[3] | **Hatam Abdoli**[1]

[1] Department of Computer Engineering, Faculty of Engineering, Bu-Ali Sina University, Hamedan, Iran

[2] Department of Embedded Systems Engineering, College of Information Technology, Incheon National University, Yeonsu-gu, Incheon, Korea

[3] Computer Engineering Department, Persian Gulf University, Bushehr, Iran

Correspondence
Mahdi Abbasi, Department of Computer Engineering, Shahid Ahmadi Roushan BLVD, Bu-Ali Sina University, Hamedan, Iran, P.O. Box 65178-38695.
Email: abbasi@basu.ac.ir

Abstract

Software-defined networks have been developed to allow the entire network to be managed as a programmable entity. As a well-known protocol in this field, OpenFlow installs new packet forwarding rules of the distinct packets of Big Data flows (known as flow entries) in the flow tables of network switches in order to implement the desired management policies. Despite the high speed, flow tables have limited capacity to store the information of Big Data flows. As a result of inefficient policy for replacing the entries of the flow table, lack of flow entries corresponding to the incoming packets in the flow table of the switch will increase the references to the controller for forwarding this packet as well as the amount of delay in packet forwarding. The underlying idea of the proposed method is to make use of the popularity of traffic flows in the table to select the intended flow for the replacement. For replacement of flow table entries, a novel and intelligent method is proposed in this research which uses a reference history of flows to assign an importance degree to each table entry. Comparison of the simulation results confirms the superiority of the method for reducing the controller's overflow.

1 | INTRODUCTION

SDN has recently become popular among researchers and industrial practitioners. Figure 1 illustrates the general structure of SDN. The main advantage of this network is its flexibility which is due to the separation of the control plane and data plane. Using software-defined networks, the entire network along with elements can be regarded as a single virtual network. Regarding this brilliant feature, the SDN is not specifically recommended for edge computing, but in practice, it can lower the complexity barriers associated with it and act as a pioneer in unleashing the true potential of edge computing. This technology helps bridge the gap between edge computing and traditional cloud combinations. For example, SDN can act as a decision-maker on whether to upload a task to the cloud or upload it at the edge for processing. The SDN controller has built-in AI that can determine when a particular link will experience high network traffic. The controller can then request further processing at the edge to eliminate the network bottleneck [1, 2].

Overall, Big Data is branded by 5Vs including volume, veracity, variety, velocity, and value. The excellent properties of SDN make it much easier to collect, send, store, and process Big Data. SDN demonstrates that it can also benefit from big data such as traffic engineering [3], cross-layer design, defence against security attacks, and SDN-based intra-datacentre and inter-datacentre networks [4]. It is shown that OpenFlow-enabled SDNs improve system performance for Hadoop-based Big Data applications by fully taking advantage of cloud infrastructure opportunities and challenges and using OpenFlow's network control capabilities to resolve network congestion [5].

These networks are controlled using certain designed software and APIs. Therefore, such networks require a communication protocol between software and hardware which could apply control software codes to all tools developed by various manufacturers [6].

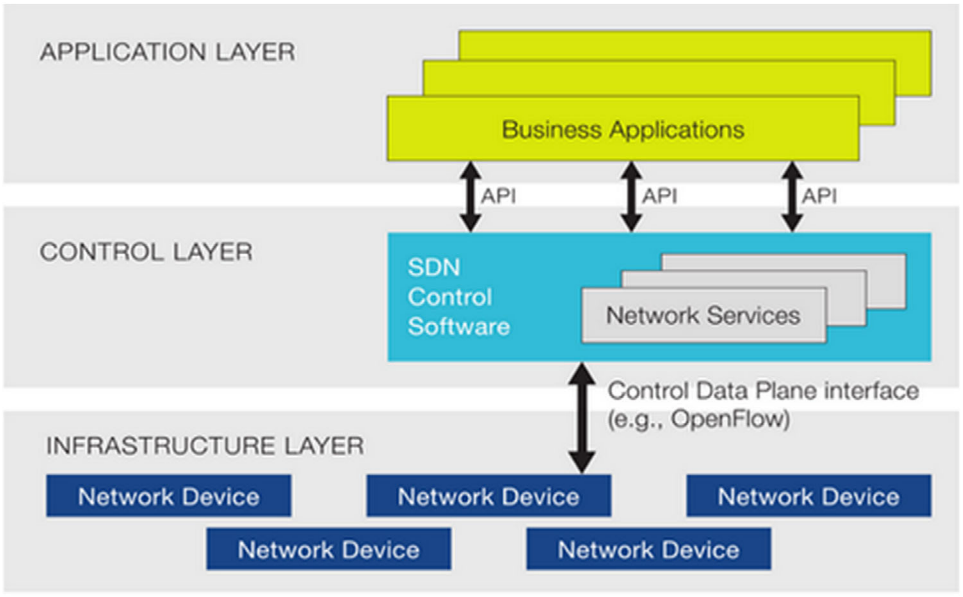

FIGURE 1 Architecture of a software-defined network

This task can be done by OpenFlow protocol. In the architecture of this protocol, that is, depicted in Figure 2, each OpenFlow switch which is responsible for forwarding network packets consists of one or more tables as well as an abstract layer. This abstract layer uses OpenFlow protocol for secure communication with the controller [7].

The most important components of each OpenFlow switch are its flow tables. Flow tables contain flow entries each of which determines how the packets belonging to a flow should be processed and sent. Flow entries are defined by the controller in flow tables [8]. These tables have limited capacity. Therefore, the number of flow entries used for storage is pre-determined and limited. Flow tables are composed of TCAMs (Ternary Content Addressable Memory). TCAMs are expensive and have a very high level of power consumption. This is why increasing the size of flow tables is costly and is likely to increase power consumption [9–11]. In a flow table, the information of each incoming flow packet is compared with the entries in the table. If it matches a table entry, the policy specified in that entry will be applied to the packet. If there is no matching entry, a message called Packet_in is issued to be processed by the controller. In this case, the processing time of the packet will increase. As a result, with increasing delay in the processing of packets, the packets of the incoming flow will accumulate in the switch's buffer and the buffer will ultimately overflow. This inefficient behaviour causes the incoming packets to be discarded at the outset. As multiple switches are connected to the controller, transmitting these messages to the controller will increase communication overload between the controller and the switch. Given this, one of the most important criteria for replacement of flow entries is to reduce the controller's overload. This is the main motivation behind the present research. As the controller is in charge of updating and replacing entries in a flow table, inefficient replacement policies will make the flow table unstable and increase the controller's computation overload. Given this important issue, the approach adopted in this study is to update the entries of the flow table based on the statistical features of their corresponding flows.

In what follows, Section 2 will review the literature and focus on the major studies which have addressed the reduction of controller's overload using algorithms for replacing the entries of the flow table. Next, we shall describe the proposed algorithm for the replacement of flow table entries to reduce the overload. Section 4 explains in detail the implementation environment of our method as well as the other methods to be compared, evaluation criteria, and comparison results. Finally, Section 5 concludes the discussion and makes suggestions for further development of the current work.

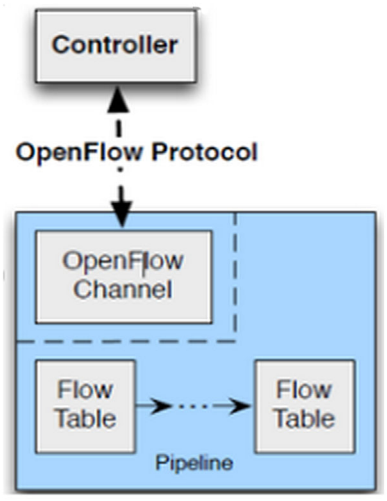

FIGURE 2 The structure of OpenFlow switch

2 | REVIEW OF LITERATURE

As mentioned above, one of the challenges ahead of software-defined networks is the limited capacity of flow tables. Given the increasing variety of applications, currently, there are more than a thousand types of flow per second in data centres. The limited capacity of flow tables in OpenFlow switches leads to increased overload in the controller because all the entries to be matched against the flows cannot be stored in it and lack of a matching entry is reported to the controller. As these tables are made up of TCAM, any increase in size will lead to higher cost and power consumption. Several solutions have been proposed by researchers [12, 13], for example, eviction techniques for removing entries from the flow table before installing new entries, compression-based techniques which decrease the accumulation of information among the flow entries of the tables as much as possible, and split-and-distribution techniques in which switches create a general distributed system whose components are dependent on each other. Eviction-based methods include replacement algorithms and mechanisms based on timeout [13]. Replacement algorithms that have so far been used in different studies and have proved to be applicable to software-based networks are FIFO, Random, and LRU.

In a 2012 study, Adam Zarek from Toronto University compared replacement algorithms (Random, FIFO, and LRU). In these algorithms, the flow table is taken merely as a cache and the entries are removed only when the table is full. Based on the mentioned replacement policies, the intended entry is selected and deleted. Zarek's study showed that FIFO is better than Random, although with a relatively slight difference. The problem with Random is that it may select those table entries for the replacement that have numerous references. LRU is better than the other two algorithms and its hit rate is higher. However, it cannot be implemented in software-defined networks. Finally, the study also examined the performance of the combination of different timeouts using the LRU algorithm. The results show that, when the length of the table is shorter than the number of active entries, the timeout size does not have a remarkable effect and any enhancement in performance depends on the replacement algorithm. The larger the size of the table, the more varied the miss rate is for different timeouts (lower miss rates for greater timeouts); after a certain point, however, the increase in table size will not affect performance [14]. In 2013, Bu-Sung Lee et al. [15] proposed a solution to reduce miss rate in flow tables and establish fairness between small and large flows in the data centre of software-defined networks. One of the features of traffic usually observed in data centres is that large flows have a very big size but are smaller in number in comparison with small flows. Therefore, large flows are more likely to be excluded due to the limited capacity of the flow table. The flooding of small flows permanently results in the exclusion of large flows from the table. As a result, when large flows arrive, the hit rate will decrease, and references to the controller increase. Thus, to overcome these issues, a cache layer is inserted between the switches and the controller. These memories can be shared by several switches. When flow entries are removed from the flow table, they are inserted into this memory. The memory contains buckets that are separately specified for small and large flows. When a flow arrives in a switch, it is first compared with the flow table. If there is no matching entry, the memory will be consulted, and if the appropriate entry is not found in the memory, the controller will be engaged. This requires extra hardware. As the size of this memory is limited, replacement algorithms will be used if the memory is full. The algorithm used in this work is LRU. When a new entry is installed in the memory, it will be replaced using LRU if the memory is full. Entries in large flows will be replaced with entries from large flows and entries in small flows with those in small flows. In 2014, Eun-Do Kim et al. developed a solution to reduce the overload of the controller of software-defined networks which is due to table miss during incoming flows. They used the LRU algorithm to manage the flow table during the replacement of flow entries. In this method, the entries are not removed when they are expired, rather they are maintained in the table as long as possible along with their age. When a flow entry is deactivated, the switch sets its counter to 0 and increments the counter of the remaining inactive entries by 1. Thus, the entry with the greatest counter is the entry recently used least. If one of the inactive entries matches the incoming packets, its timeout and counter values are reset to default and it is put among active flows. When the table size exceeds a threshold, LRU removes some of the expired entries from the table. By doing this, the hit rate of the flow table entries is relatively increased, but this will increase the number of entries in the table. The study makes use of the concept of vacancy which was added to OpenFlow switch v1.4. Thus, the switch can limit the number of inactive flow entries by vacancy-up and vacancy-down parameters and, as inactive entries remain in the table, prevent the table from overflowing. In the proposed method of this paper, the maximum number of inactive entries can be equal to vacancyUp - vacancyDown. The results have shown that this algorithm performs better than FIFO and Random [16, 17]. However, this method was implemented in OpenFlow switch while controlling tasks in software-defined networks should be implemented in the controller. Implementation of the LRU algorithm in the controller requires that the controller be informed about the last entry that matched the incoming flows, but this will impose a huge load on the controller. As this study evaluates algorithms that can be implemented in the controller and are based on software-defined networks, this algorithm will not be discussed here. Another issue to be considered is that OpenFlow controllers install flow entries with a fixed timeout in the flow table. There are some disadvantages to this. For example, in the flows with short packet intervals, if the timeout is large, the flow entry will remain in the flow table for a long time and occupy space. Conversely, if packet intervals in a flow are long and the timeout of the corresponding flow entry is small, the intended entry will be removed before the arrival of each packet of the flow, which will lead to the generation of too many packet-ins and increase the controller's overload. For this reason, Anilkumar Vishnoi et al. [18] proposed an OpenFlow controller called

Smart Time in 2014 which combined appropriate idle timeout calculation with active eviction of flow entries. They aimed to use TCAM efficiently. This was the first real implementation of a smart flow management strategy in an OpenFlow controller. Its design was based on the analysis of authentic data (retrieved from data centres) and certain clues observed in the data. For example, some flows will never repeat, or some flows have only 1–2 packets and 1 s (idle timeout in OpenFlow switches) is too much for these flows. In general, since traffic is constantly changing and the parameters of flows are different, idle timeout must be allocated according to the traffic pattern. For this purpose, on installing a flow entry for the first time, a low idle timeout is allocated to it (100 ms which has been obtained during experiments). This value ensures that short flows, as well as non-repeating flows, will not remain in TCAM for a long time. For the next repetitions of flow installation, the value of idle timeout will be calculated based on the number of repetitions:

$$\text{Idle Timeout} = \text{MinIdle Timeout} * 2^{\text{Flow Repeat Count}} \quad (1)$$

For repetitive flows, this value rises until a certain MaxIdleTimeout (10 s in this study) is achieved because it is rising exponentially and too big a rise occupies the space of TCAM. Also, the value of timeout is subtracted for short flows which repeat over long intervals. Finally, when TCAM exceeds a threshold, a flow entry will be removed from the table by use of the FIFO or the Random algorithm. In this paper, Random was selected and used. In the same year, Liang Xie et al. [19] presented a solution based on the setting of the timeout of flow entries. In their study, an adaptive control mechanism was proposed with the aim of readjusting the idle timeout using flow tables and cooperation of controller and switch. Instead of conventional methods which re-set the idle timeout of a flow entry that is activated and used, this mechanism (called Accflow) which has been used for management of the flow tables of OpenFlow switches adds the remaining idle timeout with the timeout which is allocated to an entry upon activation and takes it as the new timeout. Thus, active flows could have higher chances of remaining in the table and the mismatch rate of the table will decrease. (Idle timeout refers to the time during which a flow can remain idle and if this time ends, the flow will be removed.)

In a 2015 study, Huikang Zhu et al. [20] proposed their Intelligent Timeout Master which was aimed at calculating the suitable timeout for flows according to their features. Based on the occupation of the flow table, this model also calculates the maximum timeout to prevent overflow in the flow table. This task requires the storage of the previous information of flow entries. In 2015, He Li et al. [21] developed a storage algorithm with low complexity to achieve a higher rate of the flow table. It used prefetch and replacement techniques for predictable and unpredictable flows, respectively. The underlying idea of the algorithm is as follows: (1) When a rule is required to be installed on the switch for each flow f_i, it must be stored on all switches on the path of the flow (prefetching the rule); (2) the timer t_i is associated with the flow entry f_i, with an estimated time for the next hit with this entry. When an entry replacement takes place in a switch, the entry with the maximum timer value will be selected in the switch. Whenever the timer t_i is to be set, two types of flow are considered: predictable flows (i.e. flows coming from network disconnection services) and unpredictable flows (i.e. Automatic flows). For the former, the timer is simply set to the arrival time of the next packet. For unpredictable flows, when the estimated arrival time of the next packet is earlier than expected, the timer is updated with the interval of the arrival of the next packet; otherwise, the timer must be re-set with a doubled value (but not exceeding t-max). The evaluation criteria included the comparison of hit rate in FIFO and LRU with that in the proposed algorithm. The results suggest that the hit rate of the proposed algorithm was more than the mentioned replacement algorithms.

Zehua Guo et al. [22] proposed a method called STAR which was a routing method for software-defined networks. This method can determine the productivity of flow tables in real-time and evict the expired flows when a new flow is required. This method, too, makes use of LRU replacement and the dynamic setting of the idle timeout. Each flow entry has a tag that specifies whether it is active or inactive. When the controller installs a new flow, this tag is set to 'active' and when the last packet of the flow arrives, it is set to 'inactive'. Therefore, the controller can estimate the productivity of the flow based on active entries and remove inactive flows even before being expired. In line with this, Linlian Zhang et al. [23] proposed a method called TimeoutX that sets the idle timeout of a flow based on three parameters: the estimated duration of the flow, the type of flow, and the productivity rate of the flow table. Then, in a 2019 study, Qing Li et al. [24] developed a mechanism called HQ-Timer which is based on machine learning. This method assigns different timeout values to different flows based on the dynamicity of the traffic. It uses the Q-learning method for allocating the proper idle timeout to flows during their installation and issuing necessary commands for their eviction to improve the productivity rate of TCAM. However, the method needs a big learning dataset which requires a huge memory. In addition, the switch has limited resources and the controller's processing unit. In the same year, Abinas Panda et al. [25] proposed a method that emphasized the management of the hard timeout of flow table entries. This method dynamically assigns different hard timeouts to both predictable and unpredictable flows. For this purpose, it uses average inter-arrival time and, for eviction and replacement, it uses the LRU algorithm. One year later, Babngida Isyaku et al. [26] proposed a similar method called IHTA in which, in addition to hard timeout, the idle timeout was also set according to the traffic pattern. This too was done through inter-arrival time. The flow entry is removed from the table when there is no packet matching it at a certain time. This and other similar methods lack a mechanism for removing invalid and finished flows from the table to maintain more active networks.

Heming yang et al. [27] proposed a method (stereos) using machine learning to classify flow entries into two active and inactive classes to form an intelligent eviction strategy. In [28], D Wu et al. presented a scheme based on the LRU algorithm which can use flow table space to increase the matching rate of table entries and also raise active flow priority to reduce matching time in flow tables. In this method, when the table is full and a new entry is received, the oldest flow entry which is not matched with incoming flows for a while, will be removed. Yi shen et al. [29], proposed another management scheme (FATM) on controllers which combines the dynamic timeout and proactive eviction to manage the flow table resources. The timeout is set by a timeout assignment module according to the flow characteristics. So, the entries of short-lived flows are removed sooner than flow entries with larger packet intervals. When the space of the table is not sufficient, the proactive eviction module eliminates flow entries to prevent the table overflow. In this design, the eviction thresholds and the other parameters in both modules are set and adjusted according to the network load.

Leo Mendibourne et al. [30] developed a method for managing flow tables in software-defined vehicle access networks. This method allocates the hard timeout value based on the prediction of vehicle mobility as well as the load level of network devices. Given the relatively new emergence of the field of software-defined networks and in light of the literature reviewed above, it can be seen that none of the discussed methods were comprehensive. The techniques used in the related work are summarized in Table 1.

Only a few studies have attempted to use the attributes of flows for removing and replacing entries. In addition, only simple algorithms such as FIFO and Random have been implemented for replacement and no study has utilized dynamic methods to manage flow tables that could remarkably optimize the management of flow table entries. Therefore, our aim here is to optimize the management of flow tables by using the information obtained from the flows. In the next section, the proposed solution is explained in detail. The results of the proposed method will be compared with those of FIFO and Random algorithms.

3 | THE PROPOSED METHOD

The method proposed in this study is a novel dynamic method for the replacement of flow table entries which uses the history of reference to flows for determining the degree of importance of entries. Our method is dynamic because it is based on flow's features. It assigns a degree of importance to each flow entry according to the reference history of flows. The importance degrees are dynamically changed and updated. They are used to determine the most appropriate entry to be removed from the table. Hence, more important entries with more references in their history will remain in the table.

On installing a new entry in the flow table by the controller in the versions prior to OpenFlow 1.4, if the table is full, the switch will send a message to the controller to inform the controller about the entry not being installed due to the fullness of the table. In OpenFlow 1.4, the concept of eviction was introduced. If this feature is activated by the controller, during the installation of a new entry in a full table, one of the existing entries will be selected to be replaced based on its degree of importance. The entry with the least importance will be selected for replacement. The proposed method makes use of this feature. Thus, the importance attribute is assigned using a certain algorithm that considers the statistical features of the flow. As can be seen in Figure 3, when a flow entry is removed from the flow table for any reason and the flow_removed event is sent to the controller, the information about this flow is stored in the controller. On the other hand, when a packet enters the switch and matches none of the flow table entries, a Packet_in message containing the necessary information for forwarding the packet and installing its corresponding flow is sent to the controller. When installing the entry corresponding to this flow, it is first specified whether or not the entry has already been installed in the table. This fast search has been implemented using a hash table. If it is revealed that the flow has already been installed in the switch's flow table, its importance is calculated based on one of the proposed algorithms; otherwise, it is initialized with 1. Table 2 shows the parameters in the proposed method.

3.1 | The first proposed replacement method

In this method, three attributes are considered in determining the importance of the flows. The first attribute is F_i which is the average number of packets matching a flow in the previous time it was in the flow table. This attribute denotes the number of references to a certain flow. Its value has a direct relationship with the degree of importance such that a greater number of references means that it should remain for longer in the table and assume higher importance. The next attribute (B_i) is the average number of bytes matching the flow which, like the previous attribute, is directly related to the importance of the flow. The third attribute is the time between removing and reinstalling a flow in the table. This attribute is inversely related to the degree of importance. The reason is that if a flow does not have any reference for a long time, it is useless and should be removed to free the space of the table. This attribute is denoted by ΔT_i.

- F_i is the ratio of the number of packets (packet_count) matching the i-th flow to the duration in which the entry remains in the flow table (t_{active}).
- B_i is the ratio of the number of bytes (Byte_count) matching the i-th flow to the duration in which the entry remains in the flow table (t_{active}).
- ΔT_i is calculated as following:

$$\Delta T_i = T_{eviction} - T_{new\_install} \tag{2}$$

In Equation (2), T_{eviction} is the time in which the flow entry is removed from the table and Flow_Removed message is sent to the controller. $T_{\text{new\_install}}$ is the time in that

TABLE 1 Taxonomy table describing the related trends

Author	Proposed scheme	Used technique
Adam Zarek	The replacement algorithms (Random, FIFO, and LRU) are compared. In these algorithms, the flow table is taken merely as a cache and the entries are removed only when the table is full.	Replacement (LRU, FIFO, Random)
Bu-Sung Lee et al	Their solution tries to reduce miss rate in flow tables and establish fairness between small and large flows in the data center of software-defined networks.	Replacement (LRU)
Eun-Do Kim et al	They used the LRU algorithm to manage the flow table during the replacement of flow entries.	Replacement (LRU)
Anilkumar Vishnoi et al	They exploited an Adaptive Idle_timeout method in combination with Random replacement to manage the flow table entries.	Idle_timeout and replacement (Random)
Liang Xie et al	Their mechanism, called Accflow, has been used for the management of the flow tables of OpenFlow switches. It adds the remaining idle timeout with the timeout which is allocated to an entry upon activation and takes it as the new timeout.	Idle_timeout
Huikang Zhu et al	Their method, Intelligent Timeout Master, aims at calculating the suitable timeout for flows according to their features.	Idle_timeout
He Li et al	They developed a low complexity algorithm to achieve a higher rate in managing the flow table. It uses prefetch and replacement techniques for predictable and unpredictable flows, respectively.	Replacement (LRU, FIFO)
Zehua Guo et al	This method can determine the productivity of flow tables in real-time and evict the expired flows when a new flow is specified. This method uses the LRU replacement and dynamic tuning of the idle timeout.	Idle_timeout, Replacement (LRU)
Linlian Zhang et al	They proposed a method called TimeoutX that sets the idle timeout of a flow based on three parameters: the estimated duration of the flow, the type of flow, and the productivity rate of the flow table.	Idle_timeout
Qing Li et al	They developed a mechanism called HQ-Timer which is based on machine learning. This method assigns different timeout values to different flows based on the dynamicity of their traffic.	Idle_timeout
Abinas Panda et al	This method dynamically assigns different hard timeouts to both predictable and unpredictable flows. For this purpose, it uses average inter-arrival time and, for eviction and replacement, it uses the LRU algorithm.	Hard_timeout, replacement (LRU)
Babngida Isyaku et al	In their method, in addition to hard timeout, the idle timeout was also used according to the traffic pattern.	Hard_timeout, Idle_timeout
Leo Mendibourne et al	Their method manages flow tables in software-defined vehicle access networks. It allocates the hard timeout value based on the prediction of vehicle mobility as well as the traffic load level of the network devices.	Hard_timeout
D Wu et al	Their LRU-based algorithm uses the space of the flow table to increase the matching rate of the flow entries and to raise the active flow priorities which in turn decreases the matching time in the flow table.	Replacement (LRU)
Yi shen et al	Their management method, naming FATM, runs on the controller and combines the dynamic timeout and the proactive eviction to manage the flow table resources.	Hard_timeout

Packet_in message is re-sent to the controller for installing the new flow.

Given these three factors, the degree of importance is obtained as following:

$$\text{Importance}_i = (F_i)^q \times (B_i)^r \times (\Delta T_i)^s \tag{3}$$

In Equation (3), *q*, *r*, and *s* are the powers of the parameters and their value depends on the importance of each of these parameters. *r* and *q* are positive while *s* is negative. In this study, q and r equal +1, and s equals -1. According to this algorithm, therefore, the importance of flows with more references and repetition is higher and these flows will remain in the flow table for a longer time. Less important entries will be replaced by new ones. This method is known as importance prediction replacement (IPR) and its pseudocode is shown in Algorithm 1.

3.2 | The second proposed replacement method

The second proposed algorithm is an optimized version of the first one. The same parameters are also used in the second method. However, a history of the information about flows is maintained and the degree of importance is assigned based on this history as well as a weight function called

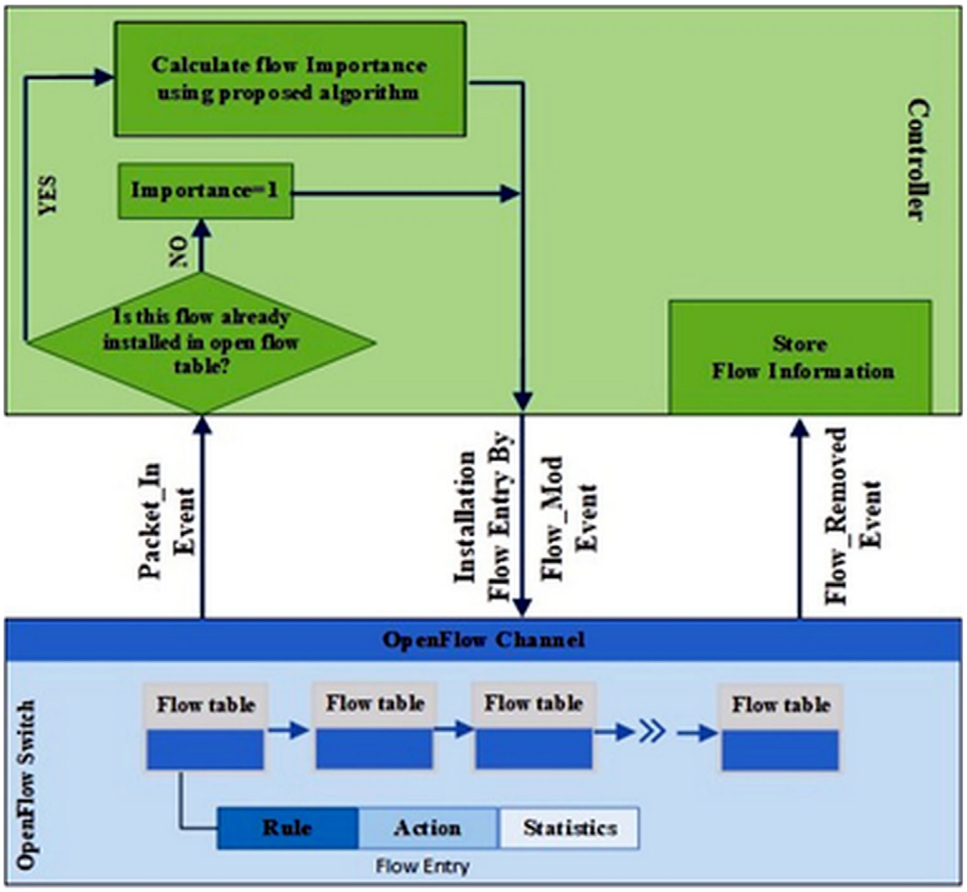

FIGURE 3 Flowchart of the proposed method

$g_i(x)$ which is based on exponential weighting moving average (EWMA).

EWMA is the simplest forecasting method that determines the amount of data at a new time point based on the average of a time interval. In this method, the data are regularly updated by replacing previous items with new ones. The number of references to previous installations of flows plays a role in calculating this value. Therefore, whenever a flow is removed from the table and Flow_Removed message is sent to the controller, the information about the flow is stored. i is the ID of the flow and x is the number of references to the controller for installing the flow. The degree of importance of the flow i in the n-th time is calculated as follows:

$$\mathrm{IMPi(n)} = \sum_{k=0}^{t} g_i(j-k), j = n-1 \tag{4}$$

Here, t denotes the sample window size. For example, if $t = 3$, only the information of the last three times where the flow has been installed in the flow table is considered in the calculation of the degree of importance. The weighting function $g_i(x)$ is a non-incremental function that increases the importance of flows that have been referred to more times and whose number of matching bytes is greater. If j is less than k, $g_i(j-k)$ is 0. This function predicts the identity of future references to the flow and the importance of each flow is dynamically measured according to the statistical information previously recorded about the attributes of the flow. Therefore, in accordance with Fang [31], we have assumed a weighting function like $g_i(x)$ which could adapt itself to changes in traffic pattern:

$$g_i(x) = \begin{cases} \frac{(1-\beta)^{j-x}}{(j-x)^{\lambda}} c_i(x), & j-t \le x \le j-1 \\ C_i(x), & x = j \end{cases} \tag{5}$$

In this equation, the closer we get to the present time (the n-th time), the greater the weight assigned to it, and vice versa. Thus, the effect of more recent values on the calculation of the degree of importance is more than the effect of more distant values.

λ is a positive number which is the power of the denominator. β ranges between 0 and 1 and equals to $\frac{2}{t+1}$ [31–33]. $C_i(x)$ is the product of the average number of packets matching the flow i in the unit of time in the x-th time and the average number of bytes matching the flow i in the x-th time. That is:

$$\mathrm{C_i(x) = F_i(x)\ B_i(x)} \tag{6}$$

TABLE 2 The parameters used in the proposed method

F_i	The average number of packets matching a flow in the previous time it was in the flow table
B_i	The average number of bytes that match the flow
ΔT_i	The time between removing and reinstalling a flow in the table
$\boldsymbol{T}_{\textbf{eviction}}$	The time in which the flow entry is removed from the table and Flow_Removed message is sent to the controller
$\boldsymbol{T}_{\textbf{new\_install}}$	The time in which Packet_in message is re-sent to the controller for installing the new flow
q, r, s	The parameters that show the importance of each of above timing parameters
i	The ID of the flow
x	The number of requests received by the controller for installing a new flow
$C_i\,(x)$	$F_i\,(x) \times B_i\,(x)$
t	This parameter denotes the sample window size
λ	A positive number which is the power of the denominator of weighing function
β	A Positive number between 0 and 1
$\boldsymbol{IMp_i(n)}$	The degree of importance of the $flow_i$ in the n-th time slot

ALGORITHM 1 Algorithm of the first proposed method

```
Events : flow_removed , packet_in
if (flow_removed(flow i)) then
    $F_i = packe\_count/t_{active}$
    $B_i = byte\_count/t_{active}$
    Save $(F_i, B_i, T_{eviction})$ in controller
end if
if (packet_in(flow i)) then
    if (first install of flow i) then
        $Importance_i = 1$
    else
        $T_{new\ install} = current\_time$
        $\Delta T_i = T_{eviction} - T_{new\ install}$
        $Importance_i = (F_i)^q * (B_i)^r * (\Delta T_i)^s\ , q = r = +1, s = -1$
    end if
    Install flow with $Importance_i$ in switch
end if
```

If x is less than 0, this value will be 0. Therefore, using Equations (4) and (5), we obtain the below equation:

$$IMp_i\ (n) = c_i\ (j) + (1-\beta)\,c_i\,(j-1) + \frac{(1-\beta)^2 c_i\,(j-2)}{2^\lambda} + \cdots + \frac{(1-\beta)^t c_i\,(j-t)}{t^\lambda} \tag{7}$$

Among the flows that enter the switch, there may be flows that repeat frequently in long intervals. Therefore, the equations should finally be combined with ΔT_i:

$$\text{Importance}_i = IMp_i(n)/\Delta T_i \tag{8}$$

The degree of importance of a flow can be calculated in this way. During eviction or replacement, therefore, less important flows are evicted and replaced earlier. This method is known as flow history-based replacement (FHR) and its pseudocode is shown in Algorithm 2.

ALGORITHM 2 Algorithm of the second proposed method

```
if (flow_removed(flow i)) then
    $F_i = packe\_count/t_{active}$
    $B_i = byte\_count/t_{active}$
    $C_i = F_i * B_i$
    Add $(C_i, T_{eviction})$ to flow__info
    if (len(flow_info) > t) then
        Remove info (1)   //" flow_ info" holds flow history up to the sample window size t
    end if
end if
if (packet_in(flow i)) then
    if (first install of flow i) then
        $Importance_i = 1$
    else
        j = len (flow_ info)
        $T_{new\ install} = current\_time$
        $\Delta T_i = T_{eviction}(last\ info\ flow\ i) - T_{new\ install}$
        $\beta = 2/t + 1\ , \lambda > 0$
        $Importance_i = c_i(j) + (1-\beta)c_i(j-1) + \frac{(1-\beta)^2 c_i(j-2)}{2^\lambda} + \dots + \frac{(1-\beta)^t c_i(j-t)}{t^\lambda}$
        $Importance_i = Importance_i/\Delta T_i$
    end if
    Install flow with $Importance_i$ in switch
end if
```

Not that all components and steps of the work have been expounded, we briefly explain the method using the flowchart of the proposed method. As can be seen in Figure 3, when a flow entry is removed from the table for any reason and the flow_removed event is sent to the controller, the information about the flow (including the attributes described in the third step) is stored in the controller. On the other hand, when a packet enters the switch and matches none of the flow table entries, a Packet_in message containing the necessary information for forwarding the packet and installing its corresponding flow is sent to the controller. When installing the entry corresponding to this flow, it is first specified whether or not the entry has already been installed in the table. This fast search has been implemented using a hash table. If it is revealed that the flow has already been installed in the switch's flow table, its importance will be calculated based on one of the algorithms described in step 4 above; otherwise, it will be initialized with 1.

4 | IMPLEMENTATION AND EVALUATION

As can be seen in Table 3, implementation was performed using Mininet [34] which is and an emulator for software-defined networks. As with the switch, OpenVswitch [35] was used which supports OpenFlow 1.4 [8, 36]. The controller used was Ryu [37] which has been written in Python and supports all versions of OpenFlow. To produce the traffic that resembles real-world traffic, we used real traffic and forwarded it on the network using TcpReplay [29].

To evaluate the method, we used two datasets called Trace_file1 and Trace_file2 for generating traffic. The sample

TABLE 3 Implementation environment

Operating system	Ubuntu 15.04
Processor	Core i3
Emulator	Mininet
Switch	OpenV Switch
Controller	RYU
Traffic forwarding tool	TCP Replay

TABLE 4 Specifications of the traffics

Dataset	Number of packets	Size of dataset	Time
Trace_file1	M11	8787 MB	1146 s
Trace_file2	3041K	1749 MB	858 s

traffics belonged to the data centres of universities [38], and are described in Table 4.

The proposed method is compared with FIFO and Random replacement methods because this method is network-based and has been implemented in the controller. Only FIFO and Random algorithms can be implemented in the controller.

The reasons why we have not compared it with LRU are as following: (1) If LRU is within the switch, it will problematize the notion of separation between the controller and the data in software-defined networks because in these networks all the control tasks should take place in the controller; and (2) if LRU is to be implemented in the controller, it needs to be constantly informed by the switch about the latest accesses to flow entries. This will lead to overload in the controller and the communication channel. Therefore, it is not correct to implement LRU in the controller [18]. Our evaluations with two tables with 500 and 750 flows and both our datasets indicate that even if the table size becomes smaller, the proposed method will still have a high performance. The maximum size of the flow table in OpenFlow switches is 2000, but here we have considered smaller sizes so that the dataset could fill the table and the results of the proposed method could be observed. The values of Hard_timeout and Idle_timeout in these experiments should be in a way that they do not affect the performance of replacement methods. (These parameters denote the maximum time an entry can remain in the table and the maximum time an entry can remain inactive in a table.) The only operator of flow table management must be the replacement method and no other element must interfere. Therefore, Idle_timeout is assumed as 0, and Hard_timeout is assumed as greater than the simulation time. Hard_timeout can be set to zero, but when both values are 0, both flows are permanent and will be never removed from the table unless the controller commands to do so. The sample window size (t) is set at 5. The evaluated parameters in this paper are as follows:

Miss_Rate: Miss_rate in the table is the most important parameter to evaluate the performance of replacement methods. Missing means that packets that enter the switch do not match any entry and are therefore sent to the controller for forwarding. As a result, the higher the miss rate, the greater the controller's overload, and vice versa.

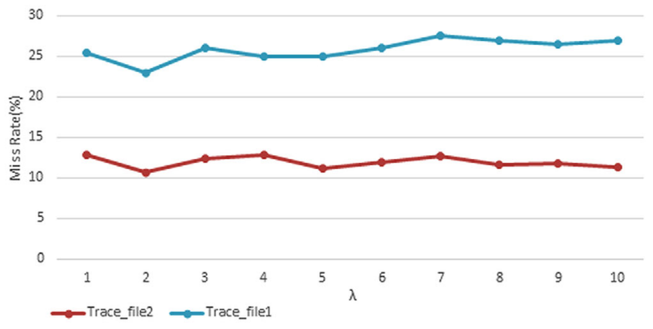

FIGURE 4 The effect of λ on the performance (miss rate of flow table) of the proposed method (with a table size of 500) to selecting the best λ. Datasets are trace_file1 and trace_file2

Miss_ByteRate (miss rate in bytes): Another parameter for evaluation of the performance of replacement algorithms is Miss_ByteRate. In the OpenFlow switch, when the packets do not match flow entries and the switch's buffer does not have enough space, the packet header along with the controller is sent to the controller. Therefore, higher Miss_ByteRate means more overload in the controller and lower Miss_ByteRate means less overload.

Calculation of the best value of λ in FHR

By doing several experiments with FHR on both datasets (trace_file1, trace_file2) and with a table size of 500, we examined the effect of λ on the performance of the proposed algorithm and selected the best λ value.

As can be seen in Figure 4, the miss rate of the flow table with $\lambda = 2$ is lower for both datasets. As a result, the value of λ was determined to be 2 for the following experiments

4.1 | The effect of the proposed method on the performance for the proposed method in terms of Miss_Rate

The miss rate of the flow table is the most important parameter for observing the reduction of the controller's overload and the performance of the proposed algorithm. Experiments 1 and 2 measure this rate for both datasets as well as the different replacement methods in comparison with IIR and FHR methods which were described in Section 3. The size of the flow table is 750 in the first experiment, and 500 in the second experiment.

As can be seen in Figures 5 and 6, the performance of the proposed method is better for both datasets. In the second experiment, the proposed method still shows a better performance after reducing the size of the flow table. On the other hand, the Random method has a better performance than FIFO in the first experiment and worse performance in the second experiment. The reason is that replacement in these methods is not based on solid logic. However, the proposed

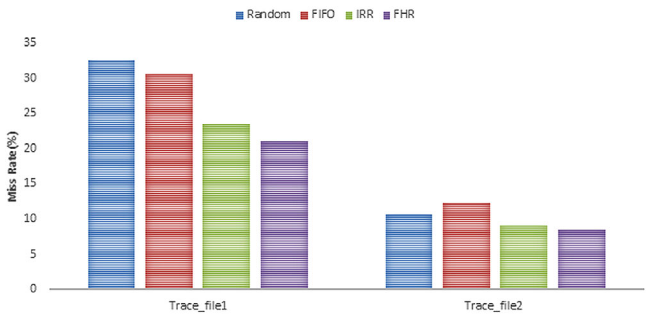

FIGURE 5 The effect of common replacement methods (Random, FIFO) in comparison with the proposed methods (IRR, FHR) on the performance (Miss Rate%) of the flow table (with a table size of 750)- this experiment done on both data sets (tarce_file1, trace_file2)

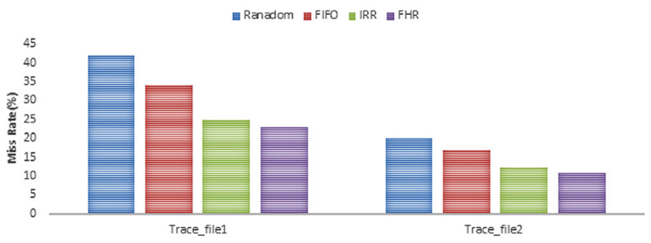

FIGURE 6 The effect of common replacement methods (Random, FIFO) in comparison with the proposed methods (IRR, FHR) on the performance (Miss Rate%) of flow table (with a table size of 500)- this experiment done on both data sets (tarce_file1, trace_file2)

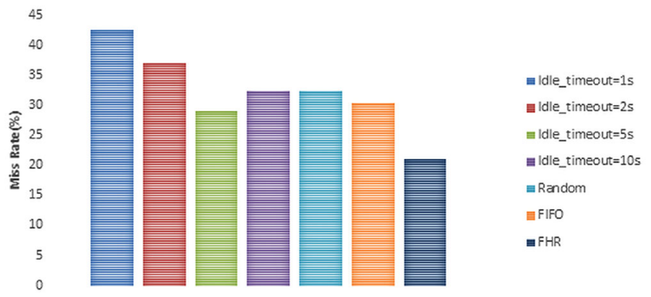

FIGURE 7 Comparison of the performance (Miss Rate% of flow table) of the proposed method with the Idle_timeout management methods, Random and FIFO (with a table size of 750). The dataset in this experiment is trace_file1

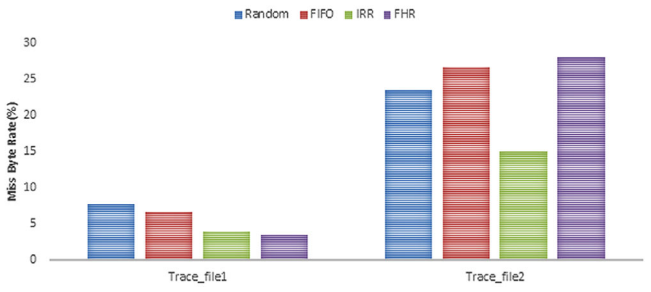

FIGURE 8 The effect of replacement algorithms on MissByte_Rate (%) in the flow table (with a table size of 750) in proposed methods (IRR, FHR) compared with FIFO and Random. This experiment done on both data sets (tarce_file1, trace_file2)

method had a higher performance under different conditions. FHR was better than IIR because it considers the history of references in calculating the degree of importance.

4.2 | Comparison of the proposed method with Idle_timeout management method

The next experiment was intended to compare the effect of flow table management using the proposed replacement method and using the allocation of Idle_timeout to the flows. The least Idle_timeout value that OpenFlow switches accept is 1 s and the maximum value according to research findings [18] is 10 s.

As can be seen in Figure 7, the proposed FHR method was compared with the Idle_timeout management method only for the Trace_file1 dataset. It is clear that the proposed method performs better than the best Idle_timeout (i.e. 10 s) in this experiment and could reduce the miss rate of the flow table. This can be compared with the results of [14, 39] which considered the values of 5 and 2–3 s, respectively, as the best value for the experimental datasets. Our proposed method, however, performed better than these two studies. The reason is that the entire space of the flow table is used in replacement methods and the flows are replaced according to attributes obtained through their history in the table.

Therefore, those flows that have been referred to most, remain in the table. Also, Figure 6 shows that, with a table size of 500, the proposed method has reduced Miss_Rate more than the methods of dynamic setting of Idle_timeout and Hard_timeout presented in [23, 25] (called TimeoutX and DHTA, respectively) which have been compared with FIFO and Random. The reason is that although setting the value of timeout can be compatible with network traffic, it cannot act adaptively and the invalid and finished flows could not be removed from the flow table in time.

4.3 | The effect of the proposed replacement algorithms on MissByte_Rate

The next experiment deals with the last evaluation parameter called Miss_ByteRate which is the miss rate of packets in bytes. When the packet information that misses the flow table is sent to the controller, if the switch's buffer has no more capacity, the entire data of the packet are also sent to the controller and create overload in the controller as well as in the communication channel between the switch and the controller. The results are shown in Figures 8 and 9 which represent a table size of 750 and 500, respectively. It can be seen that the proposed algorithms have still better performance and reduce MissByte_Rate more than other methods. The reason for reducing Miss_ByteRate in IRR and FHR is that they take into account the average of bytes matching the flow in calculating the degree of importance.

To select the suitable entry and replace the entries to update the flow table, the proposed methods make use of the

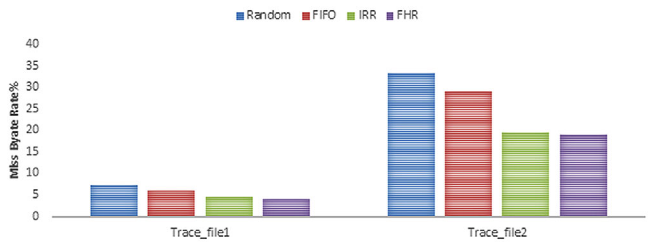

FIGURE 9 The effect of replacement algorithms on MissByte_Rate (%) in the flow table (with a table size of 500) in the proposed method (IRR, FHR) compared with FIFO and Random. This experiment done on both data sets (tarce_file1, trace_file2)

popularity of flows when they are in the table as well as the time distance between installation and removal of the flow. The implementation of the proposed method indicated that, along with FIFO and Random, it could increase hit rate and reduce the controller's overload significantly more than the other existing methods.

Therefore, to enhance the performance of the OpenFlow switch it is required to implement the proposed algorithm in the controller. On installing a new entry by the controller, the importance degree of the new flow entry is calculated and updated using the proposed algorithm. Since in the SDN networks, the controller is responsible for running all of the required operations, the switch settings remain unchanged.

In addition, the proposed method will increase the scalability in SDN networks. Scalability is considered a subject with multiple aspects in SDN. One of its aspects is the throughput. When the load of a controller decreases and the hit-rate increases, more flows can be handled in any time slot. Another aspect of scalability is reducing the delay in installing new flow entries. When the entries of flows with higher importance are included in the table, repetitive installations are not reduced. Hence, this scheme reduces the controller overhead, and increases the importance degree of highly referenced flows. Accordingly, keeping the selected flows in the table will increase the network capability for scalability.

5 | CONCLUSION

As mentioned earlier, flow tables contain flow entries each of which determines how the packets belonging to a flow of Big Data streams should be analysed and sent. Flow entries are defined by the controller in flow tables. These tables have limited capacity. They are made of TCAM memories. TCAM has a high-power consumption. Therefore, large tables are costly and have high power consumption. The incoming flows are compared with table entries and if there is no match, a message is sent to the controller for processing the flow. Thus, more time is needed for processing such a flow. If the incoming flows remain in the switch's buffer, the buffer will be filled and new packets will be discarded on arrival. As multiple switches are connected to the controller, transmitting these messages to the controller will increase communication overload between the controller and the switch. The main criterion for the replacement of the entries is the reduction of overload. The reason behind selecting this criterion is that the controller is responsible for updating flow table entries and, if entries become unstable, the controller's computation overload will increase. Lack of an entry in the table corresponding to the incoming packet will lead to reference to the controller and increase its overload.

To this end, the present paper proposed an intelligent method for replacing flow table entries to reduce the controller's overload. The focus of this study is on developing a dynamic replacement method. This intelligent method utilizes the statistical features of the traffic flows in the table to select a table for replacement and makes use of the popularity of flows in the flow table for replacing entries and updating the flow table. The method aims to evaluate the existing entries according to the history of the activities of the flow, which was neglected in previous studies. For this purpose, we used the 'importance' feature which has been introduced in OpenFlow 1.4.

Finally, we implemented the proposed mechanism in an emulator for software-defined networks and evaluated its performance in terms of several criteria. The results of the proposed method along with FIFO and Random methods show that our method could increase the hit rate and reduce the controller's overload significantly more than the existing methods. Also, this method performs better than flow management based on Idle_timeout. As mentioned in the evaluation section, this replacement method alone has a better performance than the methods of dynamic setting of timeout and will certainly bring about more successful results if combined with timeout setting. In addition, this is the first dynamic replacement method that has been implemented in the controller.

In this work, all the statistical features of flows are not used in the proposed algorithm. Hence, using all of them seems to lead to better results. For future research, this method can be implemented in real-world software-defined networks. Also, other statistical features which are obtained from the flow (for example, protocol type, flow capacity, QoS level etc.) may be used alone or in combination for developing new replacement algorithms and suitable setting of Idle_time in the flow table. Since more details of flow information will be considered, the accuracy and quality would be improved and the flows with higher hit probability would remain in the table. All these techniques can be combined to obtain better results.

CONFLICT OF INTEREST

No

FUNDING

None

DATA AVAILABILITY STATEMENT

Data available on request due to privacy/ethical restrictions

REFERENCES

1. Dinh, P.T., Park, M.: BDF-SDN: A big data framework for DDOS attack detection in large-scale SDN-based cloud. In: 2021 IEEE Conference on Dependable and Secure Computing (DSC), Fukushima, Japan (2021)

2. Chu, X., et al.: Big data and its V's with IoT to develop sustainability. Sci. Program. 2021 3780594 (2021)
3. Abbasi, M., et al.: Efficient flow processing in 5 G-envisioned SDN-based Internet of Vehicles using GPUs. IEEE Trans. Intell. Transp. Syst. 22(8), 5283–5292 (2021)
4. Montaño, M.: IoT management analysis using SDN: Survey. Applied Technologies, Springer, Berlin (2021)
5. Tarek, A., et al.: Software-defined networks towards Big Data: A survey. Advanced Machine Learning Technologies and Applications, Springer, Berlin (2021)
6. Kreutz, D., et al.: Software-defined networking: A comprehensive survey. Proc. IEEE 103(1), 14–76 (2015)
7. McKeown, N., et al.: OpenFlow: Enabling innovation in campus networks (OpenFlow White Paper). http://www.openflowswitch.org (2008). Accessed 15 November 2021
8. <openflow-spec-v1.4.0.pdf>
9. Sezer, S., Scott-Hayward, S., Kaur Chouhan, P., Fraser, B., Lake, D., Finnegan, J., Vilijoen, N., Miller, M., Rao, N.: Are we ready for SDN? Implementation challenges for software-defined networks. IEEE Communications Magazine 51(7), 36–43 (2013)
10. Abbasi, M., et al.: Ingredients to enhance the performance of two-stage TCAM-based packet classifiers in internet of things: Greedy layering, bit auctioning and range encoding. EURASIP J. Wireless Commun. Networking 2019(1), 1–15 (2019)
11. Vakilian, S., Abbasi, M., Fanian, A.: Increasing the efficiency of TCAM-based packet classifiers using dynamic cut technique in geometric space. J. Adv. Def. Sci. Technol. 6(1), 65–71 (2015)
12. Alsaeedi, M., Mohamad, M.M., Al-Roubaiey, A.A.: Toward adaptive and scalable OpenFlow-SDN flow control: A survey. IEEE Access 7, 107346–107379 (2019)
13. Nguyen, X.-N., et al.: Rules placement problem in OpenFlow networks: A survey. IEEE Commun. Surv. Tutorials 18(2), 1273–1286 (2016)
14. Zarek, A., Ganjali, Y., Lie, D.: Openflow timeouts demystified. Computer Engineering Research Group: University of Toronto (2012)
15. Lee, B.-S., Kanagavelu, R., Aung, K.M.M.: An efficient flow cache algorithm with improved fairness in software-defined data center networks. In: 2013 IEEE 2nd International Conference on Cloud Networking (CloudNet), San Francisco (2013)
16. Kim, E.-D., et al.: Flow table management scheme applying an LRU caching algorithm. In: 2014 International Conference on Information and Communication Technology Convergence (ICTC), Busan, Korea (2014)
17. Kim, E.-D., et al.: A flow entry management scheme for reducing controller overhead. In: 16th International Conference on Advanced Communication Technology (ICACT), Pyeongchang, Korea (2014)
18. Vishnoi, A., et al.: Effective switch memory management in openflow networks. In: Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, Mumbai (2014)
19. Xie, L., et al.: An adaptive scheme for data forwarding in software defined network. In: Sixth International Conference on Wireless Communications and Signal Processing (WCSP), Chennai (2014)
20. Zhu, H., et al.: Intelligent timeout master: Dynamic timeout for SDN-based data centers. In: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, Canada (2015)
21. Li, H., et al.: FDRC: Flow-driven rule caching optimization in software defined networking. In: 2015 IEEE International Conference on Communications (ICC), London (2015)
22. Guo, Z., et al.: STAR: Preventing flow-table overflow in software-defined networks. Comput. Networks 125, 15–25 (2017)
23. Zhang, L., et al.: TimeoutX: An adaptive flow table management method in software defined networks. In: 2015 IEEE Global Communications Conference (GLOBECOM), San Diego, CA (2015)
24. Li, Q., et al.: HQTimer: A Hybrid ${Q} $-Learning-Based Timeout Mechanism in Software-Defined Networks. IEEE Trans. Network Serv. Manage. 16(1), 153–166 (2019)
25. Panda, A., et al.: Dynamic Hard Timeout based Flow Table Management in Openflow enabled SDN. In: 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN), Vellore, India (2019)
26. Isyaku, B., et al.: IHTA: Dynamic idle-hard timeout allocation algorithm based OpenFlow switch. In: 2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE), Malaysia (2020)
27. Yang, H., Riley, G.F., Blough, D.M.: STEREOS: Smart table EntRy eviction for OpenFlow switches. IEEE J. Sel. Areas Commun. 38(2), 377–388 (2019)
28. Wu, D., Qiao, L., Chen, Q.: Research and implementation of LRU-based flow table management for onboard switch. In: 2020 Prognostics and Health Management Conference (PHM-Besançon) (2020)
29. Shen, Y., et al.: AFTM: An adaptive flow table management scheme for OpenFlow switches. In: 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS) (2020)
30. Mendiboure, L., Chalouf, M.A., Krief, F.: Load-aware and mobility-aware flow rules management in software defined vehicular access networks. IEEE Access 8, 167411–167424 (2020)
31. Fang, C., Huang, T., Liu, J., Chen, J.-y., Liu, Y.-j.: Fast convergence caching replacement algorithm based on dynamic classification for content-centric networks. J. China Univ. Posts Telecommun. 20(5), 45–50 (2013)
32. Montgomery, D.C., Johnson, L.A., Gardiner, J.S., Forecasting and Time Series Analysis. McGraw-Hill Companies, New York (1990)
33. Kachru, U., Production & Operations Management. New Delhi, Excel Books (2009)
34. Mininet: An instant virtual network on your laptop (or other PC). http://mininet.org/. Accessed 20 January 2021
35. Open vSwitch: an open virtual switch: http://openvswitch.org/. Accessed 20 January 2021
36. Specification-Version, O.S., 1.4. 0. Open Networking Foundation (2013). Accessed 20 January 2021
37. https://osrg.github.io/ryu/-ryu-controller. Accessed 20 January 2021
38. Sekaran, R., et al.: Survival study on Blockchain based 6 G-enabled mobile edge computation for IoT automation. IEEE Access 8, 143453–143463 (2020)
39. Metter, C., et al.: Analytical model for SDN signaling traffic and flow table occupancy and its application for various types of traffic. IEEE Trans. Network Serv. Manage. 14(3), 603–615 (2017)

How to cite this article: Abbasi M., Maleki, S., Jeon, G., Khosravi, M.R., Abdoli, H.: An intelligent method for reducing the overhead of analysing big data flows in openflow switch. IET Commun. 1–12 (2022). https://doi.org/10.1049/cmu2.12328.