electronics

MDPI

Article

Adaptive and Hybrid Idle–Hard Timeout Allocation and Flow Eviction Mechanism Considering Traffic Characteristics

Babangida Isyaku [1,2,*], Kamalrulnizam Abu Bakar [1], Mohd Soperi Mohd Zahid [3] and Muhammed Nura Yusuf [1,4]

[1] Department of Computer Science, Faculty of Engineering, Universiti Teknologi Malaysia, Johor Bahru 81310, Malaysia; knizam@utm.my (K.A.B.); ymnura@atbu.edu.ng (M.N.Y.)
[2] Department of Mathematics and Computer Science, Sule Lamido University, P.M.B 048 Kafin Hausa, Jigawa State, Nigeria
[3] Department of Computer and Information Sciences, Universiti Teknologi PETRONAS, Seri Iskandar 32610, Perak, Malaysia; msoperi.mzahid@utp.edu.my
[4] Department of Mathematical Sciences Abubakar Tafawa Balewa University, P.M.B 0248 Bauchi, Bauchi State, Nigeria
* Correspondence: isyaku@graduate.utm.my or bangis4u@gmail.com; Tel.: +60-1161-110-962

Received: 20 October 2020; Accepted: 20 November 2020; Published: 23 November 2020

Abstract: Software-defined networking (SDN) enables flexible fine-grained networking policies by allowing the SDN controller to install packet handling rules on distributed switches. The behaviour of SDN depends on the set of forwarding entries installed at the switch flow table. The increasing number of traffics from the proliferation of the Internet of Thing (IoT) devices increase the processing load on the controller and generates an additional number of entries stored in the flow table. However, the switch flow table memory (TCAM) cannot accommodate many entries. Packets from multimedia flows are usually large in size and thus suffer processing delay and require more flow set up requests. The SDN controller may be overloaded and face some scalability problems because it supports a limited number of requests from switches. OpenFlow uses timeout configuration to manage flow setup request. The conventional fixed timeout cannot cope up with the dynamic nature of traffic flows. This paper controls the frequent flow setup requests by proposing an adaptive and hybrid idle–hard timeout allocation (AH-IHTA). The algorithm considers traffic patterns, flow table usage ratio, and returns appropriate the timeout to different flows. The performance evaluations conducted have shown a 28% and 39% reduction in the flow setup request and flow eviction, respectively.

Keywords: SDN; OpenFlow; flow table; TCAM; adaptive; hybrid timeout; dynamic traffic pattern

1. Introduction

Software-defined networking (SDN) has recently gained popularity among researchers, industry, and even carrier-grade network due to simple and programmable network management. These merits come because of a new computing paradigm that improves the complexity of the conventional network protocols and removes the vendor-specific instruction of network devices (i.e., switch or routers). This can be achieved through the separation of planes, and the fact SDN decouples the control plane (networking logic) from the data plane (forwarding logic) [1]. OpenFlow is used as the standard communication interface to separate the control from the data planes [2]. The SDN controller mainly decides how the switch will process every incoming flow in real-time through the installed flow handling entries on distributed switches' flow tables. End-point policies will determine the source to destination switches for each flow while the shortest path will be determined by routing policies

according to the routing algorithm [3]. For example, the shortest-path routing policy asks the OpenFlow switches to forward packets along the shortest path between two nodes. Incoming packets of every flow are processed according to the flow table entries' configuration setting either by reactive or proactive approaches. In the former, whenever packets arrive at switches during network operation, its corresponding flow entry will be installed into the switch flow table by the controller on demand. For the latter, flow entries are installed in the switch flow table in advance before the packet arrives. In most, cases for both approaches, the switch may consult the controller when matching entries are not available, and issue a packet-in event to update the flow table with the new entry that allows the packet to reach its destination [4]. Although there are several reasons for which *packet_in* events are sent to the controller, a proactive approach may usually have a smaller number of packet-in generation events [4].

The switch flow table is commonly implemented with special high-speed memory ternary content addressable memory (TCAM) technology. Regardless of the number of stored entries for traffic flow, finding an entry match or not will take the same amount of time [4]. These features are highly commendable since, despite the speed, it can further support parallel lookup for both exact and wildcard match rules with constant O (1) time. However, TCAM is power-hungry, expensive, and available in a limited space that can only accommodate from 750 to 20,000 flow entries [5,6]. A prior study reported that in large-scale networks there can be up to 200 k flow arrival per second [5]. An average flow size is relatively small with at least 20 packets per flows and the inter-arrival time is quite high in less than 30 ms [3].

The recent growth of multimedia services due to the proliferation of the Internet of Thing (IoT) devices had significantly increased the amount of traffic flows in the internet, and this causes the traffic pattern to exhibit different variability in terms of arrival and duration [6]. This consequently may surge the processing load on the controller because of the need to process every incoming flow and install the corresponding flow entries in the switch flow table. In addition, these concerns increase the level of packet-in generation due to inappropriate timeout and an ineffective flow entry eviction policy. Therefore, this leads to overhead concern, and because of the constraints of central processing unit (CPU) processing power, memory, bandwidth, and other resources, the controller can only process a limited number of the received packet-in event [7]. The flow table switch memory limitation and the huge load on the controller result in a scalability problem [6], owing to the contradiction that fine-grained policies require a large number of flow entries to be stored in the flow table, while the current TCAM flow is insufficient to accommodate the required flow entries.

The preliminary design of the proposed research in this paper has been presented in [8]. However, the previous work did not consider the classification of data flow based on protocols such as transport control protocol (TCP), user datagram protocol (UDP), and Internet Control Message *Protocol* (ICMP). This paper complements the previous work by extending the previous experiment to consider various data flow classification and more eviction mechanisms were added to show the effectiveness of the proposed evictions mechanism.

1.1. Timeout Mechanism

Open flow configures flow table entries with idle and hard timeout across the flows [9]. These timeouts are used to control the life span of flow entries in the switch flow table. When the time elapses, the flow entry is removed to free spaces for new incoming entry. An idle timeout removes the correspondent entries after an inactive period with no packet to match that entry, while a hard timeout removes the corresponding flow entry after a deterministic lifetime of the entry, regardless of whether it has been matched or otherwise. In the conventional method, the SDN controller configured flow entries with the fixed idle or hard timeout value across the flows without considering the variabilities exhibited by the flows and usage of the flow table [9]. The work of [7,9] is an example of such a strategy under a specific condition and application, respectively. This method could be effective with better performance when the traffic flows are fixed and flow table space is sufficient. However, assigning the

same timeout value results in the inefficient utilization of switch flow table memory because it may be too long for short lived flows or too short for large long lived flows [10]. As such, this exhibits a trade-off between the short and large timeout value. The former may prematurely evict a flow entry before the arrival of the next packet while the latter may cause a larger flow table than necessary [10]. Consequently, it causes the switch to controller communication overhead by the sending of a *packet-in* message to the controller when the next packet arrives. Conversely, a longer timeout reduces premature evictions, but come at a cost of flow table occupancy [10]. A larger timeout value lengthens the flow entry lifetime even though there is no packet expected to match that entry. For example, in Figure 1, the small square shapes represent the number of packets that have been transmitted for each flow (F) until time t1, t2. F1 represents a short flow with a small number of packets at both t1 and t2, while F2 shows long flow with a small number of packets and F3 illustrates short flow with a large number of packets. Finally, F4 has a long flow with large packets.

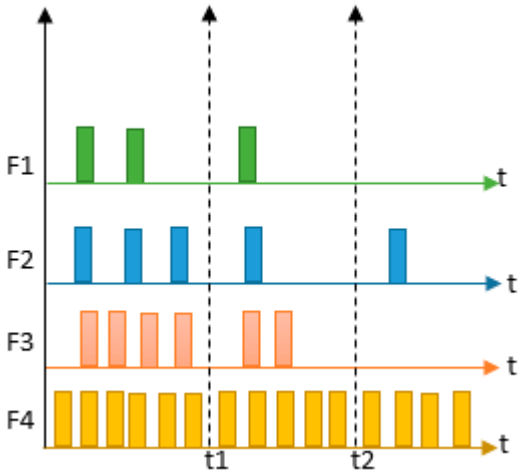


Figure 1. Example of flow with variabilities.

Assigning fixed timeout from F1 to F4 may cause the switches to frequently consult the controller to install an entry corresponding to F3 and F4 at a slight time interval because of the nature of their large packets, while F1 and F3 may occupy the space with few packets to match the entries. Therefore, it is required to investigate the feasibility of an adaptive and hybrid flow timeout strategy that can incorporate fixed and dynamic timeout values based on observed natures of intra-flow packet inter-arrival time. Some proposals were made in [10–12] to assign flexible timeout. However, they only consider idle timeout. However, the work of [13] employed a hard timeout. Only considering either idle or hard timeout is not enough to reduce the communication overhead and improve the efficiency of the limited flow table. The idle timeout may be too small for large flows and hard timeout may truncate flows during data transmission. More needs to be done to flexibly apply both timeouts without having to pay for their drawback. For example, in Figure 1, a flow like F1 and F2 can be overcome with small idle timeout value because they live in a short period with a small number of packets at a longer inter-arrival time, while F4 lives for a longer period with a large number of packets at a short inter-arrival time. Obviously, it will be effective to assign a long hard timeout value to F4 but may not be worthwhile to assign a hard timeout to F1. For F2, the duration is also short but the inter-arrival time is a bit greater than F1 flows, as such a problem can also be dealt with an idle timeout mechanism without assigning idle-timeout equivalent to F1. The earlier version of this work considered an idle and hard timeout. In this research, it was further extended to consider a different data flow type, such TCP, UDP and other protocols, because of the large data packets that are being generated by these protocols in addition to the data flow inter-arrival time. Afterwards, when the flow table reached its maximum full capacity, in contrast to the conventional flow eviction policy explained earlier, an efficient eviction mechanism was employed to remove unused flow entries.

1.2. Eviction Mechanism

When the flow table is full to its capacity, OpenFlow allows the provision of an eviction mechanism to recover the memory space occupied by less important or inactive rules [6]. To enable eviction in OpenFlow, one can learn from the conventional rule replacement used in computer memory management. When the flow table of switches is not enough to accommodate new flows, the SDN controller uses first-in-first-out (FIFO), random, and least recently used (LRU) concepts to actively remove less important or inactive rules like in the work of [14,15]. A prior study by Zarek et al. [16] verified that LRU outperformed the other eviction method with better performance in terms of hit ratio. However, the main challenges with all the eviction methods were deciding on the right rule to evict. FIFO, on the one hand, is an ineffective eviction method because generally, there are many flow processed packets in a short period of time. As such, FIFO has a tendency to preserve a rule for processing less frequently to stay for a longer period with a low hit ratio, occupying a precious space. Another way is to remove flows randomly, but one can attest that the method can remove rules that correspond to the active flows. This problem is not peculiar to FIFO and random alone, as LRU is not an exception as well. Suppose 200 different flows arrived at the switch with a different number of packets each at a slight time interval (t). The switch flow table space is less than the number of arrived flows, assuming the available space is for 100 flow entries. In this situation, the eviction mechanism is enabled to evict 100 less important entries. Figure 2 illustrates the processing of inefficient eviction methods. F_ID represents the unique ID of each flow, and time shows the matching time of entry, P_count represents the number of packets corresponding to each flow. Suppose at time t_0 2 flows F1 and F2 arrive at the switch with 10 and 16 packets, respectively. Among the flows, there are delay-sensitive and best-effort flows. Delay sensitive flows include multimedia flow such as video or voice and the best effort is like email and other web traffic flows. Assuming delay-sensitive flows are frequently used, this type of flow needs to be prioritized because of the nature of their stringent delay requirement. Flow F1 and F2 represent delay-sensitive flows. At time t_1, 10 more packets arrive from flow F1, one more packet arrives from flow F2 and three flows arrived (F3–F5) with 10, 6, and 3 packets respectively. Assuming a proactive approach is used, a match is performed, and action is taken accordingly without consulting the controller. Similarly, at time t_2, more packets belonging to F3–F5 arrive. However, at time t_3, two flows—F6 and F7—arrive. Unfortunately, the flow table is already full to its capacity, thus, some flows have to make way for the newly arrived flows. Assuming two flows are needed to be evicted, applying the LRU scheme in this scenario caused flow F1 and F2 to be the victims. It is important to note that, the number of packets determined an important flow entry. F1 has the highest number of packets followed by F2. Therefore, at time t_4, the previously removed F1 and F2 arrive. In this situation, the switch must trigger a packet-in event to request for corresponding entries. Consequently, besides the communication overhead, packets belonging to F1 and F2 must suffer another processing delay and thus this affects the system quality of service (QoS).

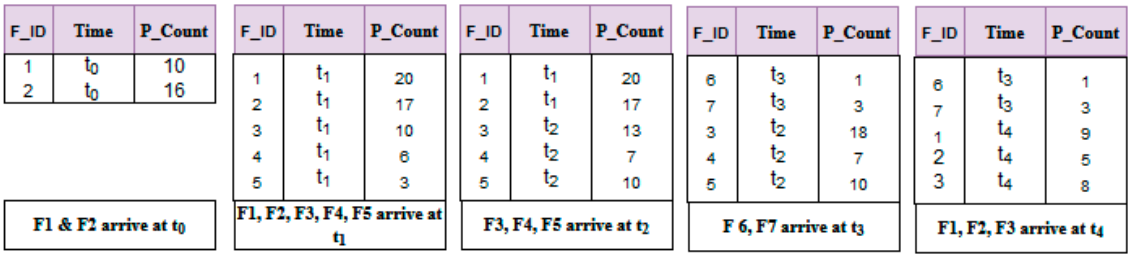


Figure 2. Example of inefficient flows eviction scheme.

Clearly, flows with the smallest number of packets should be the right candidates to evict, which are F4 and F5, not F1 and F2. The selection of flows to be evicted should be based on the number of packets in the corresponding flow. This criterion is used in our research to evict less important flow entries from the flow table when it is full.

The paper road map is organized as follows: Section 2 presents flow timeout and eviction related work. Section 3 explains the variabilities exhibited by flows. Afterward, Section 4 describes the design of the proposed solution. Section 5 presents the system evaluation and experiment using the Ryu controller and OpenVSwitch. Finally, Section 6 concludes the paper.

2. Related Work

Deciding a suitable timeout value is quite challenging [17], as an inappropriate timeout value can cause flow table overflow and increases the communication overhead on the controller. Therefore, several timeout allocation logics have been proposed in Lu et al., [10], Xu et al., [4,11,18,19] to assign a flexible timeout value. These studies, choose to adjust the timeout value based on flow state, and switch location in some cases decides to recycle flow entry for a second chance without the issuance of a *packet-in* message. These approaches have improved the flow table utilization and reduced the need for the controller to explicitly remove an entry. However, the allocation logic relies on idle timeout which may not be suitable for long lived flows with a large number of packets [8]. In addition, the implementation logic requires the controller to obtain the accurate knowledge of flow statistical information, since the idle timeout value is usually small (5–9 ms) [16]. Therefore, large signalling overhead to gather statistics at high frequency is unavoidable [20]. In contrast, another method proposed in Panda et al. [13] focused on adaptation for the hard timeout value considering predictable and unpredictable flows. Their work aimed to preserve the predictable flows because of their importance and reduce the occupancy rate of unpredictable flow because of their spontaneous nature. Once the flow table is full, they activate the eviction policy based on LRU. The work of [21] proposed an adaptive idle timeout based on ICMP, TCP, and UDP flow classification, where different values are assigned efficiently. However, flow may experience a large packet processing delay before installation because of the need to pull the statistics of flows from the flow table. This may not be a suitable solution for a delay-sensitive application. A dynamic hybrid timeout method is proposed in [22] to secure the flow table against distributed denial of service (DDoS) attacks. In their work, a large idle timeout value was set to flows with longer duration while a small hard timeout was assigned to flows with short duration when the flow table usage was close to capacity. This method has reduced the problem of overflow to some extent. However, it may not give optimal performance with long lived flows with short packet inter-arrival time due to its nature of a fixed timeout setting. Similar work proposed in [23] addressed rule dependency. Instead of relying on a hard timeout, flows that experienced entry-miss were configured with an idle timeout value along with dependent flows, and persistent rules along with proactive eviction using FIFO methods were used. However, the time allocation schemes used either idle or hard timeout. In our research, adaptive and hybrid timeout allocation schemes were used considering a different dataflow, flow table utilization ration, flow duration and packet interarrival time.

Despite assigning a flexible timeout value, the presence of a large number of flows reported in [24] filled up the flow table more quickly. Thus, suitable eviction mechanisms are important to enable the removal of unused flow entries efficiently. Several eviction implementation logics are proposed in [14,15,24,25] and [26] to eliminate entries of lower importance to free space. Intuitively, these approaches implement first-in-first-out (FIFO) [23], random [19], and least recently used (LRU) [13,25] logic. Others rely on machine learning techniques [24]. To circumvent flow table overflow and overhead, Kim et al. [27] proposed short flow first (SFF) replacement scheme considering flow features. The scheme leverages on the matching period of flow entry and determines a subject of the flow entry replacement with the features. To some extent, SFF has improved the efficiency of the flow table and reduced the overflow problem. However, they neglected the incorporation of an adaptive timeout to different flows based on the protocol type which may introduce overhead as the traffic flows over time. While research on eviction schemes have shown some improvements, more need to be done to improve the flow table utilization and reduce the overhead mainly caused by frequent flow setup requests, due to a poor timeout setting and eviction policy. The main challenges with the

conventional method remain ineffective because of failure in identifying the proper less important entry to be evicted [28]. Ideally, eviction algorithms should performed based on the lifetime and importance of flow entry, and it is set by enabling the corresponding flags in the configuration of the OpenFlow switch. Therefore, in OpenFlow, eviction logic is implemented as an optional feature in the flow table data structure, and LRU, machine learning techniques, and other schemes may not necessarily be supported by some OpenFlow switches [20].

In contrast to the existing schemes, our proposed method considered the adaptation of hybrid idle and hard timeout without having to pay for their respective drawbacks. Moreover, the timeout values were also adjusted based on the flow table utilization ratio, the variabilities exhibited by flows, and different data flow types, such as TCP, UDP, and ICMP. Unused flow entries are evicted based on the importance of flows which will be obtained through packet count built-in data collection in OpenFlow.

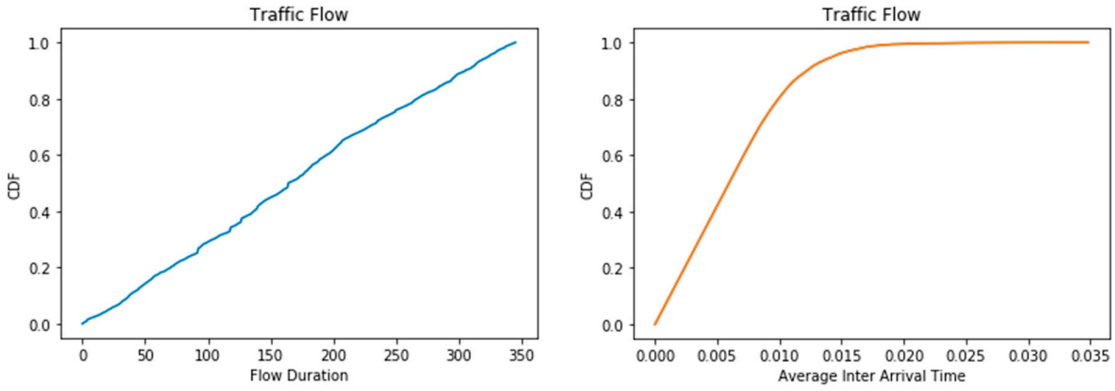
3. Traffic Flow Pattern

Traffic flows are shown to exhibit variabilities in terms of duration and inter-arrival time. To verify such variabilities, publicly available packet trace data from University centres was analysed to study the distribution of flow duration, packet inter-arrival time, and the data flow type. This will also help to answer the question of why flexible timeout should be promoted than static timeout to reduce communication overhead; secondly, the question of which flows will benefit from long hard timeout and short idle timeout values to reduce the significant flow table space occupancy for less used flows.; and thirdly, which flows will be evicted when the flow table capacity reaches its maximum capacity to prevent evicting frequently used flows with a large number of packets. The real traffic traces UNI1 dataset used in [29] from the campus data centre is composed of traffic generated from a number of varying applications and different protocols. Although a different definition of flow exists in the literature, the size of flows is mostly determined by flow rate. The rate is the number of packets or bytes count generated by the application during a fixed time interval. In our research, the flow is defined as the stream of packets sharing five (5) tuples (source and destination IP address, source and destination port number, and protocol) subject to timeout. Flow duration is the difference between the first and the last packet of the flow. The data flow type represents the type of protocol (TCP or UDP flow, etc.). Figure 3 illustrates the cumulative distribution of flow (CDF) duration and the average inter-arrival time of two consecutive packets belonging to a flow.

It could be observed that the duration of flows differs greatly and ranges from a minimum value of 1 s to a maximum of 350 s. The size of flows is mostly defined by the packets count fixed threshold value. The work of [23] classifies flows (e.g., 1–2 packets as small; 2–10 packets as a medium; and more than 10 packets as large flow). Therefore, considering that it is unknown whether there will be a second packet when the first packets of the flow arrives, a small idle timeout is sufficient for small flows, which will significantly reduce the flow table occupancy and overhead. A larger hard timeout is quite enough for a larger flow to ensure that the flow is never removed during packet transmission. In addition, the large timeout can also prevent multiple flow requests for large flows with longer duration. For the inter-arrival time, as explained previously, the conventional fixed timeout value across flows may not fit with the flow behaviour. One of the greatest problems is that for the flow inter-arrival time, as illustrated in Figure 3b, different flows exhibit different inter-arrival time. More than 70% of the flows are having an inter-arrival timeout of 0.010. Configuring flows with the fixed value may cause a large flow installation request which in turn generates high communication overhead. Therefore, in the proposed design, this shortcoming is overcome with an adaptive small value for every first flow. This will help to study the behaviour of new flows, as flows that repeat more often their timeout are adjusted accordingly.

It was observed that most of the traffic flows are generated by TCP, UDP, and ICMP packets. Table 1 show the three data flow protocols and the number of packets belonging to each. There is a large number of packets in the trace, however, some flows contain a single packet. This submission is

not peculiar to this paper. Other network researchers have noticed that the majority of traffic flows are very small [30], although some flows are large, which carry a considerable portion of the total packets, which was similarly observed from our analyses. The ratio of the TCP traffic flows is quite high, which amounts to 85%. One could verify that most of the packets are generated by TCP after UDP flows. The majority of UDP flows have only single packets, which amounts to 10% of the total packets. Conversely, ICMP has the lowest number of packets, 3%. These analyses served as a guide to the paper to decide on the following; (1) which flow to assign higher priority with higher initial timeout value, and which flow to evict when the flow table reached its maximum capacity to prevent tempering with important flows.



(**a**) (**b**)

Figure 3. Flow duration (**a**) and inter-arrival time (**b**). (**a**) Distribution of flows duration. (**b**) Distribution of average flow inter-arrival time.

Table 1. Statistics of a packet from different data flows from the dataset.

Data Flow	Statistics	
Protocol Type	**Packets**	**Percentage**
TCP	3,895,012	85.87%
UDP	469,807	10.36%
ICMP	171,306	3.77%

4. Design of the Proposed Adaptive-Hybrid Idle and Hard Timeout Allocation Algorithm (AH-IHTA) and Flow Eviction

This paper proposes an adaptive-hybrid idle and hard timeout allocation (AH-IHTA) scheme to improve the efficiency of the flow table and reduce the controller overhead without modifying the SDN architecture. In achieving this goal, AH-IHTA focuses on setting an adaptive timeout to different flows according to data flow duration, inter-arrival time, data flow protocol type, and flow table capacity rather than a fixed timeout across flows.

AH-IHTA is implemented as an application on top of the Ryu controller. As shown in Figure 4, this consists of four (4) functional modules: (1) flow table capacity monitoring module (FCMM), (2) data flow installation module (DFIM), (3) adaptive-hybrid idle and hard timeout allocation (AH-IHTA), (4) data flow eviction module (DFEM). The general workflow of the AH-IHTA starts with the controller obtaining the flow table capacity (total number of active flow entries in the residence) from switches at a regular time interval t and stores the data in FCMM. When a new data flow arrives or table miss-entry occurs, the controller extracts the flow table capacity value from FCMM and verifies it with the flow table usage (number of flow entries required). Depending on the flow table usage, if the usage is high, DFEM is called to remove the data flow with less packet count to free up space for the next incoming data flow. Then, DFIM is called to install the corresponding new flow entries. If flow table usage is low

or medium, DFIM is directly called to install the corresponding new data flow entry. The following subsections described the four (4) modules in detail.

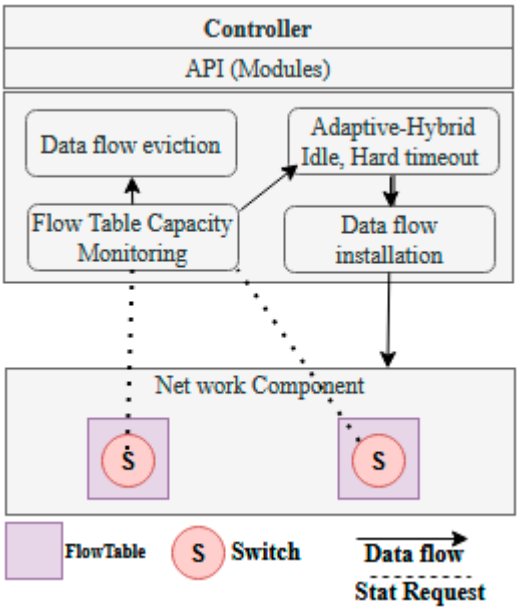


Figure 4. Modules of the proposed solution.

4.1. Flow Table Capacity Monitoring Module (FCMM)

FCMM aims to record the accurate flows statistical information and resource usage in recent times with minimal overhead. This information will be used to return an appropriate hybrid and adaptive timeout value to different flow behaviours. In addition, when the flow table is full of its capacity, such information will help in deciding the right flows to evict. OpenFlow protocol provides extensive built-in data flow statistics collection to support different applications. One can leverage on EventOFPAggregateStats, an event trigger feature provided by the Ryu controller using a thread monitor mechanism to collect the statistic of data flow in the switch flow table at a different time interval. However, when a flow lasts longer than the statistics polling interval, obtaining such statistics at the regular interval required an extra processing load. This process can potentially be resource-intensive for the SDN controller and introduce another scalability concern especially when the traffic flows increased and the number of switches to be monitored increases as well. Experimentally, it has been observed that when the flow idle timeout value (∇_t F_{IT}) is smaller than the polling monitoring interval, this results in a number of packet-in events due to the miss-entry of short lived flows (SLF). Alternatively, the flow idle timeout value (∇_t F_{IT}) may be set to a value greater than the polling monitoring interval (PMI). However, when the flow idle value (∇_t F_{IT}) is greater than the PMI value, it results in potential flow table occupancy longer than necessary by long lived flow (LLF) with few or no packets. On the other hand, when the PMI value is short, then it increased the communication overhead between the switch controller, which in turn defeats the objective of the proposed solution. Therefore, completely depending on the controller to pull the data of total active flows to obtain the usage of the flow table may not be the best solution. Thus, the proposed design does not completely depend on the polling flow statistic to obtain flow information. Instead, three statistical data were introduced to relieve the overhead for setting timeout values and flow eviction: AH-IHTA uses two different sources of information to decide on the appropriate timeout value: a database is maintained in the FCM module, which maintains the information of the active flows in every switch. When the SDN controller receives a packet-in event, a corresponding FlowMod is sent to install an entry. Once a flow is installed, it will be stored in the database using the following format {(*dpid*, *cookie*): (*idle* and *hard timeout*): (*match*, *time*)}. *dpid* identifies the switch and *cookie* is the unique value to identify each flow, whilst the *idle* timeout for

each entry and total *hard* timeout are recorded. The *match*ing header field and the timestamp when an entry is installed are also recorded. This database module will not only reduce the overhead due to the frequent polling statistic counter but will also help the controller to process a new packet-in event much faster. A capacity threshold value is used to obtain the number of active entries with least overhead. This way, the current flow table utilization is obtained through the proportion of total active entries at sampling time. The FlowRemoved flag was configured to obtain the number of packets corresponding to a particular flow and its duration. The duration and packets_count is used to check the flow volume for proper categorization as illustrated in Figure 5. Based on average flow duration, if the duration of flow f is greater than 11 s and the packet_count is greater than 10, such a flow is categorized as a long lived large flow (LLLF). When the average duration is less than 2 s and the packet_count is also less than 2, then it is categorized as short lived small flow (SLSF). The remaining flows whose average duration is within 2–11, with more than two packets but less than 10, are categorized as medium lived flows (MLF). The time interval difference of two consecutive flows can be extracted from the time recorded in the flow database. The packets matching period between these flows is used to categorize the flow packet inter-arrival time (PIAT). The matching period represents the packet inter-arrival time belonging to the same flow. δ represents the time interval between the first and last packet of the flow. Therefore, when a new packet matches an existing entry, the time is recorded as (∇t_i), and the maximum flow waiting time is denoted as δ.

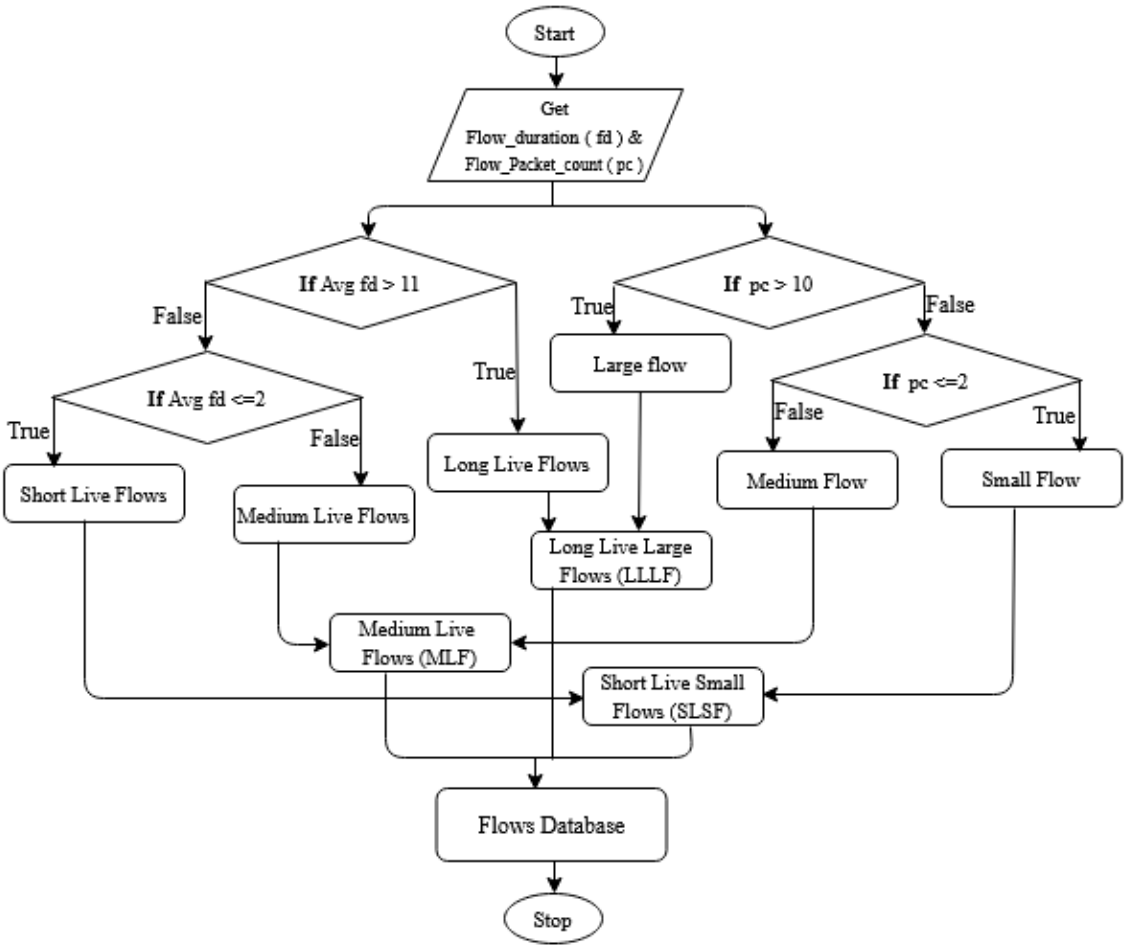


Figure 5. Flow categories.

4.2. Adaptive-Hybrid Idle and Hard Timeout Allocation Algorithm (AH-IHTA)

The procedure of AH-IHTA at the initial stage lets $F = \{f_{i.....}f_n\}$ denote a set of n flows coming into the switch; a flow $f_i \in F; with\ i = 1, 2 \ldots n$. f_i is defined as a stream of packets having the same

five (5) tuples (source and destination IP, source and destination port number, and protocol) subject to timeout Δ. A corresponding entry (e_i) for each flow f_i is installed in the switch flow table. Entry (e_i) is used to match the header field against the packet and action to process the packet is set, and a priority (i) indicates the precedence among the (e_i). A bigger (i) represents higher priority. Therefore, since different packets may arrive from different protocols which may have a different requirement, this way, without a loss of generality, all (e) have a unique priority and action depending on the data flow protocol. The entire data flow entry sets are donated as {E}, while the data flow entering the switch belonging to the internet protocol (IP) are denoted as $E_{\Phi.}$ Other data flows are denoted as $E_{\beta.}$ The operator ∈ explains the relationship between the incoming packets and the data flow protocol. P ∈ E_i means that packet p belongs to the data flow entry E_i. Note that upon the arrival of packets in a switch, it will be classified as to whether its data flow contains IP packets or not. P ∈ M e_{ip} is used to denote packets p matching e_{ip}, where e_i has the highest priority $\{e_j|P \in e_j\}$.

Thus, when the packet arrives with no corresponding entry, a table miss will occur. A switch generates a packet-in message to the controller. The controller application called a packet-in handler to extract the information about the packet for further processing. $P_i \boxed{\leftrightarrow} E_\Phi, j$, represents the association between the miss-entry and the data flow protocol type. E_T, E_U, E_I, represent TCP, UDP, and ICMP data flows, respectively, as illustrated in Equation (1), while E_O represents other data flows, as shown in Equation (2). Equation (3) presents the whole data packets that may arrive from a different protocol into the switch flow table. Note that these packets may arrive at the same time interval or time t2 slightly after t1, with large or few packets:

$$\forall\, i, j.\ p_i\ (E_T, E_U, E_I,) \in E_\Phi \tag{1}$$

$$\forall\, i, j.\ p_i\ (E_O) \in E_\beta \tag{2}$$

$$E_\Phi\ U\ E_\beta \in F_i \tag{3}$$

Packet p_i and entry E_j are associated with each other, or $p_i \boxed{\leftrightarrow} E_j$ either in E_Φ or E_β, when the following condition is satisfied:

$$p_i \cap E_j\ U\ \{p_j|p_j \rightarrow pi\} = \emptyset \tag{4}$$

The association in Equation (4) indicates for all arrived packets *pi*, of entry *j*, belong to the IP data flow entry, and the p_i entry match is not found in the flow table. One of the reasons for the unavailability of the entry is mostly attributed to the unavailability of flow table space occupied by less important entries due to inefficient timeout settings. Therefore, to set an appropriate timeout, the current design considered the data flow type, flow table utilization ratio, packet inter-arrival time, and flow duration. Flow table utilization ratio is categorized into: low (L), medium (M), and high (H). This way, the proportion of real-time active flow entries in the flow table is used to obtain the utilization ratio. The ratio parameter is considered as an input for adjusting the lifetime of entries when the flow table utilization changes due to an increase in traffic flow.

$$\varphi = \frac{E}{C} \times 100\% \tag{5}$$

Equation (5) presents the ratio (φ) of the active flow entry in the flow table at a different time interval. E indicates the total current active flow entries at time (∇t_i) and C presents the total capacity of the flow table storage. The controller obtains the real-time total active flow entries through the flow table capacity monitoring module explained previously. Part of the objective is to use the packet inter-arrival time (PIAT) to return an efficient timeout value. However, initially, at the network booting timeout, it may be difficult to obtain the PIAT. Therefore, every new incoming packet will be classified based on their protocol and the initial ($\nabla_t F_{IT}$) will be returned according to the protocol requirement. As for subsequent packets, AH-IHTA checks conditions such as PIAT and flow table utilization ratio stored in FCMM to return the suitable ($\nabla_t F_{IT}$) and ($\nabla_t F_{HT}$), respectively. Flow idle timeout value (∇_t

F_{IT}), since it is usually a small value, and many flows in the network also stay for a short period. In this method, the current (φ) will be check as illustrated in Equation (6). If the current (φ) is less than or equal to 25%, the usage is considered to be low:

$$(\varphi_{curr} \leq 25\%) \text{ and } (p < p_{\text{th}}) \in \text{Flow Table}_{\text{L}} \tag{6}$$

The data flow packet is further checked for whether it belongs to TCP, UDP, and ICMP. If the packets of flows satisfy the condition in Equations (7)–(9), a minimum flow idle value (∇_t F_{IT}) is set to 1, 3, and 5 s for ICMP, UDP, TCP, respectively with a priority value. The initial value was derived from the average min and max hard timeout value for each packet and the OpenFlow standard only accepts an integer as the timeout value. As for the priority, to make it simple, a fixed priority value can be assigned across flows. However, this idea may not be good for connection-oriented data flow. For example, TCP is a reliable transmission protocol with a connection of the transmission layer. Therefore, fixed priority across all packets may affect TCP performance by disrupting existing flows. While, on the other hand, UDP and ICMP are protocols without a connection layer. Thus, assigning a priority value slightly lower than E_T may not affect their performance:

$$\forall\, p \in E_{\Phi} \text{ and } \{E_T | p \in E_T\} \tag{7}$$

$$\forall\, p \in E_{\Phi} \text{ and } \{E_U | p \in E_U\} \tag{8}$$

$$\forall\, p \in E_{\Phi} \text{ and } \{E_I | p \in E_I\} \tag{9}$$

When the network state is in the condition illustrated in Equation (6). There are some old flows, and increasing (∇_t F_{HT}) to a maximum value of 11 s may prolong the life span of such flows. Of course, this improves the utilization of the flow table, because of the low usage. However, since traffic flows exhibit variabilities, it may easily change from LLLF to short-lived small flow (SLSF) with little or no packets. This can easily overflow the limited flow table and block new traffic flow which may consequently lead to packet drop because of the unavailability of space. Alternatively, the eviction policy may be activated to remove such flow entries as early as possible. However, the early activation of flow eviction policy may introduce another overhead of frequently replacing the old flow entry with the new ones. To circumvent the overhead, an additional condition is set to check if the utilization changed to Equation (10). The overhead changes with respect to the flow table utilization and traffic flows. Since the number of active flow entries increases, the traffic flow evolves, and this situation requires adjusting for the flow entries' survival lifetime. Therefore, the challenge can be overcome by re-adjusting to reduce (∇_t F_{HT}), based on PIAT, the duration, and the data flow protocol type. Thus, the hard timeout of (SLSF) with a maximum value will be reduced. The summary of flows' configurable timeout value is presented in Table 2:

$$(\varphi_{curr} > 25\% \text{ and } \leq 50\%) \text{ and } (p \leq p_{\text{th}}) \in \text{Flow Table}_{\text{M}} \tag{10}$$

However, when the flow interval and packet inter-arrival time (PIAT) increases, the flow table occupancy rate may easily change. In this situation, the proportion of the total active entries will significantly increase too. Therefore, a total flow table capacity threshold is required to balance the occupancy rate and overhead, since it may not be flexible and efficient to set additional timeout or install additional entry. Moreover, the process of adding an entry (update operation) in the switch flow table is hard and very slow when the flow table is full to its capacity. As the capacity threshold is required to enable an eviction policy to remove unused entries as illustrated in Equation (11), an experimental prior study reports that when the flow table usage ratio reaches from 80% to its capacity, it may not give optimal performance [15]. Therefore, once the threshold is crossed, the eviction policy will be activated. Due to the poor nature of a flow table update operation, a certain amount of free space needs to be preserved (20%) to speed up the process, and strictly control the increment in entry lifetime.

This way, the AH-IHTA module set the T_{del} flag to notify DFEM to explicitly remove a less important entry considering the traffic characteristics: AH-IHTA is presented in Algorithm 1.

$$(\varphi_{curr}>50\ \% \leq 85\%) \text{ and } (p > p_{\text{th}}) \in \text{Flow Table}_{\text{H}} \quad (11)$$

Table 2. Summary of the flow configurable timeout values of idle–hard timeout allocation (IHTA).

Timeout	Type	Description	Value
∇_1	Idle	Idle timeout for every new arrived flow at time t_1 and the usage of flow table is <25%	Initial value 1, 2, and 3 s for IP (E_Φ) such as ICMP, UDP, TCP flows respectively
∇_2	Idle	Idle timeout for subsequent flows when the flow table usage is >25% but <50%	For E_Φ $\nabla_2 > \nabla_1$, and default value for non-IP (E_O). E_O value is the average between min and max (∇_1) of E_Φ
∇_3	Idle	Idle when the flow table usage > 50% and <85%	For E_Φ $\nabla_3 < \nabla_2$ default value for E_O
∇_4	Hard	Idle timeout for every new arrived flow at time t_1 and the usage of flow table is <25%	For E_Φ $\nabla_4 \leq$ Max and default value for E_O
∇_5	Hard	Long lived flows with a large number of packets when the flow table usage is >25% but <50%	For E_Φ $\nabla_5 < \nabla_4$ default value for E_O
∇_6	Hard	Long lived flows with a large number of packets greater than (N) when the flow table usage > 50% and ≤ 85%	$\nabla_6 < \nabla_5$

Algorithm 1. Adaptive-Hybrid Idle and Hard Timeout Allocation Algorithm (AH-IHTA).

1. **Procedure** AH-IHTA (Flow Table capacity (FC), Packets (P), Threshold (N))
2. **Output**: AH-IHTA timeout
3. *packet_count* → 0
4. New flow arrival **then**
5. Check IP data flows ∀ i, j. p_i (E_T, E_U, E_I,) ∈ E_Φ
6. **If** data flow → IP data flows **then**
7. $p_i \cap E_j$ U { p_j | $p_j \rightarrow pi$} = ∅
8. Check *Flow Table U ratio* in *FCMM* **then**
9. **If** ($\varphi_{curr} \leq$ 25%) and ($p < p_{\text{th}}$) ∈ Flow Table_L **then**
10. FIT ←*initial*
11. **elif** *(φ_{curr} >25% and ≤ 50%) and ($p \leq p_{th}$) ∈ Flow Table_M* **then**
12. **FIT** ←Set
13. FHT ←Set
14. elif (φ_{curr}>50% ≤ 85%) and ($p > p_{\text{th}}$) ∈ Flow Table_H **then**
15. Adjust FIT ←Set
16. Adjust FHT ←Set
17. **else**
18. *DFIM* install an entry with FIT → DEFAULT
19. *DFIM* install an entry with FHT→ DEFAULT
20. **return AH**-IHTA
21. **END**

4.3. Data Flow Installation Module (DFIM)

DFIM module uses the OpenFlow application programming interface (API) to install and modify data flow entries according to the priority and timeout value return of AH-IHTA. For all new incoming flow, it set an initial (∇_t F_{init}) and unique priority to different IP packets. The installation time for each entry is recorded. Thereafter, when the behaviour of traffic flow change or subsequent flows arrived, it adaptively adjusted the (∇_t F_{IT}) and (∇_t F_{HT}), respectively, with respect to the flow table utilization ration and the recorded inter-arrival time and duration. The unique required parameters for this module include the switch datapath_ID, flow entry cookie, the message incoming port, the return idle or hard timeout by AH-IHTA module, flow priority, the other five tuple fields to match the flow, and

the command to specify the operation. The five tuple fields determine the flow matching operation through the OFPMatch object provided by the Ryu controller. Thereafter, data flow encapsulation will be carried out according to the parameter passed into OFPFlowMod using OFPFC_AD command action to add flow entry.

4.4. Data Flow Eviction Module (DFEM)

Once the threshold value illustrated in Equation (11) is crossed, the DFEM module is executed to free up space for the next incoming flows. Although at sampling time DFEM module checks for non-IP flow with no packets and removed them explicitly. As for the IP data flows, the DFEM module checks packet counts and flow with the maximum (∇_t F_{HT}). A flow with a fewer packets count is considered to be a small flow with longer inter-arrival time. Therefore, such flows with maximum (∇_t F_{HT}) are the victim and therefore, are removed by the DFEM module as illustrated in Algorithm 2. Afterward, the FlowRemoved message will be sent to the controller for notification, and the FCMM counter will be updated accordingly. This will not only save storage space, but it will also reduce the extra packet processing delay, especially to a delay-sensitive application data flow.

Algorithm 2. Data Flow Eviction (DFE).

```
1. Procedure FEE (Flow Table capacity (FC), Packets (P), Threshold (N))
2. Output: FEE
3. FEE at sampling time (t)
4. Check data flows
5. If data flow → non-IP data flows (∀ i, j. p_i (E_O) ∈ E_β) then
6.    delete entry explicitly
7. Else
8.     If Flow Table capacity is full then
9.      Check data flow → Check IP data flows ∀ i, j. p_i (E_T, E_U, E_I,) ∈ E_Φ
10.       Foreach (IP data flow in the Flow Table)
11.          check packet_count
12.          If packet_count ≤ (N)
13.              Select entry I with FHT_Max
14.              DFEM → delete an entry I
15.              Send FlowRemoved (I) to the controller
16.              FCMM update counter
17.   return DFE
18.   END
```

5. Experimental Result and Discussion

Simple network topology was employed to test and evaluate the proposed solution as illustrated in Figure 6. Iperf utility is considered as a traffic test mechanism to generate TCP and UDP data packets with various networking parameters such as bandwidth, interval, and protocol. One host is configured as a server, as traffic is generated between the server and clients. Different instances were considered, firstly, single TCP and UDP traffic at the reporting interval of 10 s with a total test duration of 30 s were generated. Secondly, parallel connections were made with 10 hosts lasting for 30 s. On average, the packets are generated with a size of 50 bytes. Afterward, the result was captured to study and analyse. 1. The number of packets in a messages event due to flow idle and hard timeout. 2. The eviction mechanism, for which flows are evicted due to the flow table capacity threshold.

The experimental results of the proposed solution presented in terms of the number of packet-in generation events due to the flow set up request, as a result, assigning fixed timeout across flows. Secondly, the packet count is considered as the parameter that indicates the important flow entry. Normally, the frequently used traffic flows exhibit a higher packets count. Such flows should not be the victim to be removed. In this way, the proposed eviction solution is compared with random, FIFO, and

LRU. All the scenarios are tested with respect to the changing timeout value, flow table size, and an increasing number of hosts.

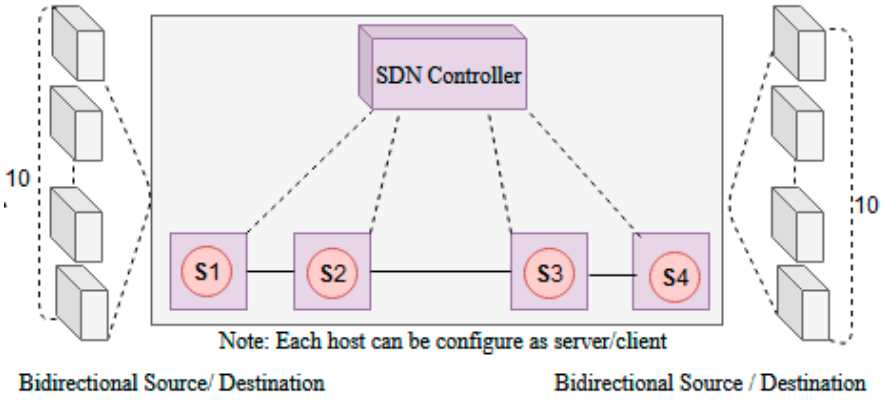


Figure 6. Network topology configuration.

5.1. Result for Idle and Hard Timeout

Figure 7a compares the performance of the proposed scheme (AH-IHTA) with different static timeout values, during the simulation process network traffic, as generated on average for 10 min. A total of 600 flows was generated with a fixed flow table size. The average number of active flow entries was captured and analysed together with the average number of packet-in generated in every second. Initially, a single connection was set with 2 s as a reporting interval and the flow table utilization was low. In this situation, the number of packet-in generation was stable. However, when the connection was changed to a parallel connection, there was the same reporting time interval. The number of packet-in generations surges with respect to the number of flows. In addition, from the gathered simulation statistics, it was further observed that most packets are generated by TCP flows, which is consistent with packets traced and analysed earlier. Despite the surging of the packet-in event as a result of the increase in flows and crossing the flow table threshold value, AH-IHTA has a lower number of packet-in generation compared to the static method. The merit is attributed to adaptive and hybrid logics for different data flows.

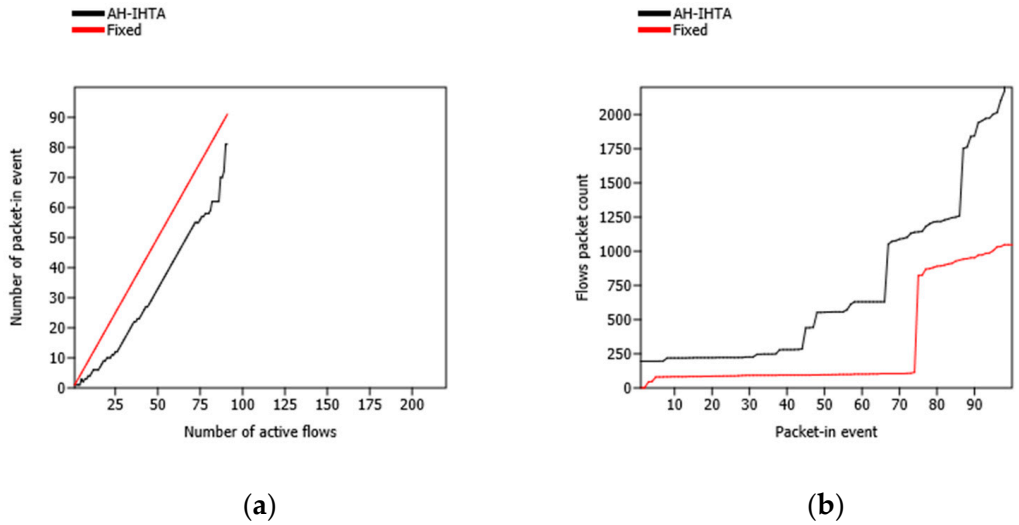
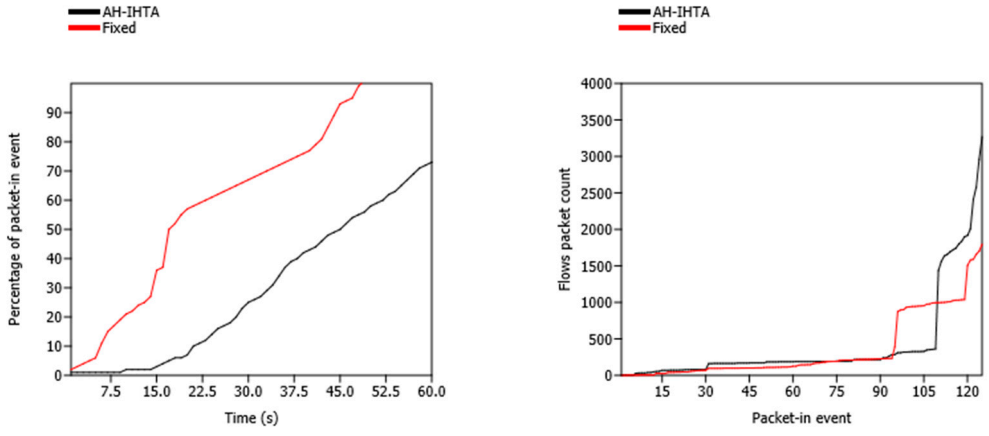


Figure 7. Results of idle and hard timeout with fixed flow table size. (**a**) Number of packet-in event with respect to number of active flows. (**b**) Flows packet count with respect to packet-in event.

Since it was observed that flows exhibit variabilities with long and short lived flows. Long lived flows usually contain a large number of packets and one of the concerns of this paper is to preserve such flows. In addition, they can also be used to describe the occupancy rate of the precious flow table. This way, Figure 7b illustrates how AH-IHTA preserved such flows. as shown from Figure 7b, initially during a single connection, as different UDP and TCP were generated with different reporting intervals. Few packets were generated, and the entry lives in the flow table for some seconds, but when the connection status changed to parallel, the number of packets for entry surged. In the case of a static method, several entries with large packets became a victim. The challenges were attributed to the fixed timeout nature of the timeout setting which cannot favour a long life with large packets. The effectiveness of AH-IHTA adaptive logic favours long lives with a large number of packets, especially during parallel connection. This way, the aim of preserving such traffic is achieved.

5.2. Result for Idle and Hard Timeout with Different Flow Table Size

The number of flow generations was increased, and the flow table size was also increased to accommodate 2000 flows. Figure 8a shows the percentage of packet-in generation events generated per minute due to the flow set up request. The static mechanism showed an upward trend with an average 68% packet-in generation within 30 s while AH-IHTA has lower packet-in generation on average 28%. Therefore, this result verifies that AH-IHTA has better performance in terms of lower packet-in generation. Similarly, in Figure 8b, it was observed during the simulation that five of the hosts send heavy-load packets flows to another host. In this situation, the static timeout mechanism prematurely expired the timeout for those flow entries and therefore, prioritized the flow with few packets. AH-IHTA considers packet the count before setting an appropriate timeout value, as flows with large packets are given priority. This is to ensure that flows with few packets never occupy the precious flow table space. Therefore, AH-IHTA has better flow table utilization.



(**a**) (**b**)

Figure 8. Results of the idle and hard timeout with different flow table size. (**a**) Percentage of packet-in event with respect to time (s). (**b**) Flows packet count with respect to packet-in event.

5.3. Result for Data Flow Entry Eviction

Figure 9 presents the eviction result when the flow capacity crossed the eviction threshold value. Flow table capacity was set with 100 and 200 flows, respectively, and the 85% capacity threshold was set accordingly. A large number of traffic flows were generated to test and verify the results considering different eviction algorithms (FIFO, least recently used, random).

From Figure 9a, it can be observed that LRU evictions were close to 78%, random 70%, FIFO close to 60%, and the proposed DFE 39%, based on the flow packets count. This way, LRU has the highest number of flow eviction, a high large number of packets followed by random and coming

down to FIFO. The proposed solution has a lower number of evictions. Thus, the conventional eviction mechanism will be expensive for delay-sensitive applications. In addition, they can further incur a high processing load because of the frequent premature eviction of active flows. The result is further verified in Figure 9b by increasing the number of traffic flow generations and increasing the flow table size. LRU evictions were close to 88%, random 58%, FIFO 62%, and DFE 47%. In both the scenarios, the proposed DFE had a lower percentage of eviction based on the flow packet counts compared to the existing method.

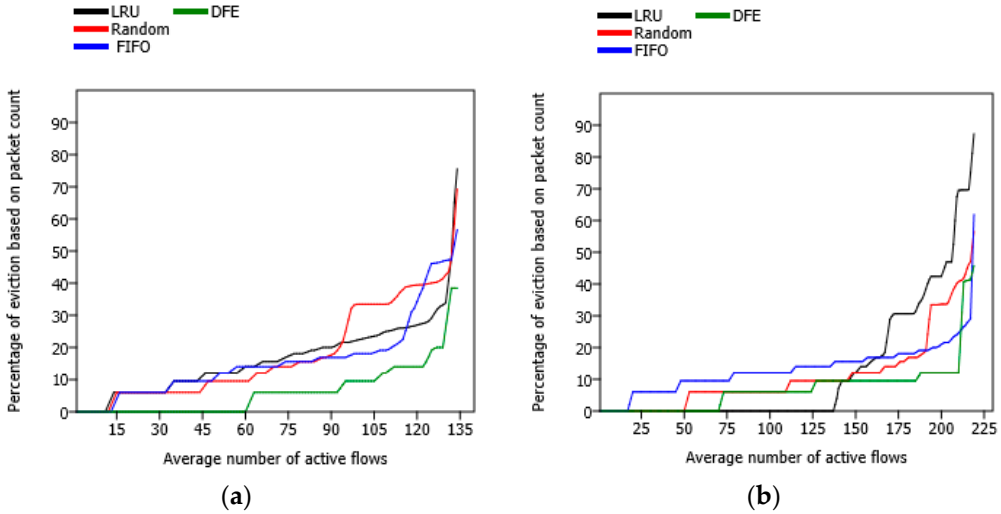


Figure 9. Flow eviction result with fixed and different flow table size. (**a**) Percentage of eviction with respect to number of active flows. (**b**) Percentage of eviction with increase in traffic flows, and flowtable size.

6. Conclusions

This paper moves a step further to improve the efficiency of the switch flow table thereby reducing the number of packets in a message to the controller. Although several efforts were made in the past to improve the controller scalability. Our work differs from the existing works by the novel classification of data flows and adaptively configuring flows with different hybrid idle and hard timeout values based on flow table utilization ration, data flow type, and packet inter-arrival time. In addition, once the traffic flows crossed the threshold value, flows with fewer packets are considered as the most appropriate eviction candidate. Such flows are evicted to free up space for new incoming flows. This is achieved by leveraging on OpenFlow build-in data structure (packet-count) at every sampling time T before evicting an entry. However, experimentally, it has been observed that it is not feasible to obtain the exact packet count in real-time before the flow finish. However, it is still effective for classifying the less important flows to be evicted compared to the conventional eviction methods. Experimentally our proposed approach AH-IHTA with DFE eviction has reduced the number of packets-in messages on average by 28% and reduced flow eviction by DFE 39%, with preserved long lived flows with a larger number of packets. Therefore, the proposed solution will be a good candidate for delay-sensitive applications where a large number of switches are required to be managed by a single controller.

Author Contributions: Conceptualization, B.I.; methodology, B.I., M.N.Y.; writing—original draft preparation, B.I.; writing—review and editing, M.S.M.Z., M.N.Y.; supervision, K.A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Ministry of Education Malaysia and Universiti Teknologi PETRONAS (UTP) through the grant number (FRGS/1/2018/ICT01/UTP/02/4).

Acknowledgments: We thank Universiti Teknologi Malaysia (UTM) for providing the environment for conducting this research work. In addition, one of the authors of this paper would like to also thank Tertiary Education

Trust Fund (TETFund) and Sule Lamido University, Kafin Hausa, Nigeria, for generous support to pursue his postgraduate studies.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Alsaeedi, M.; Mohamad, M.M.; Al-roubaiey, A.A. Toward Adaptive and Scalable OpenFlow-SDN Flow Control: A Survey. *IEEE Access* **2019**, *7*, 107346–107379. [CrossRef]
2. Open Networking Foundation. *SDN Architecture Overview*; ONF: Palo Alto, CA, USA, 2013; pp. 1–5.
3. Zhang, S.Q.; Zhang, Q.; Tizghadam, A.; Park, B.; Bannazadeh, H.; Boutaba, R.; Leon-Garcia, A. TCAM space-efficient routing in a software defined network. *Comput. Netw.* **2017**, *125*, 26–40. [CrossRef]
4. Kim, T.; Lee, K.; Lee, J.; Park, S.; Kim, Y.; Lee, B. A Dynamic Timeout Control Algorithm in Software Defined Networks. *Int. J. Future Comput. Commun.* **2014**, *3*, 331–336. [CrossRef]
5. Yan, B.; Xu, Y.; Chao, H.J. Adaptive Wildcard Rule Cache Management for Software-Defined Networks. *IEEE/ACM Trans. Netw.* **2018**, *26*, 962–975. [CrossRef]
6. Isyaku, B.; Soperi, M.; Zahid, M.; Kamat, M.B. Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey. *Future Internet* **2020**, *12*, 147. [CrossRef]
7. Liang, H.; Hong, P.; Li, J.; Ni, D. Effective Idle timeout Value for Instant Messaging in Software Defined Networks. In Proceedings of the 2015 IEEE International Conference on Communication Workshop (ICCW), London, UK, 8–12 June 2015; Volume 2015, pp. 352–356.
8. Isyaku, B. IHTA: Dynamic Idle-Hard Timeout Allocation Algorithm based OpenFlow Switch. In Proceedings of the 2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE), Penang, Malaysia, 18–19 April 2020; pp. 170–175.
9. Zhang, L.; Lin, R.; Xu, S.; Wang, S. AHTM: Achieving Efficient Flow Table Utilization in Software Defined Networks. In Proceedings of the 2014 IEEE Global Communications Conference, Austin, TX, USA, 8–12 December 2014; pp. 1897–1902.
10. Lu, M.; Deng, W.; Shi, Y. TF-IdleTimeout: Improving Efficiency of TCAM in SDN by Dynamically Adjusting Flow Entry Lifecycle. In Proceedings of the 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Budapest, Hungary, 9–12 October 2016; pp. 2681–2686.
11. Xu, X.; Lin, H.; Fan, Z. An Adaptive Flow Table Adjustment Algorithm for SDN. In Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications, IEEE 17th International Conference on Smart City, IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 10–12 August 2019; pp. 1779–1784.
12. Li, Z.; Hu, Y.; Zhang, X. SDN Flow Entry Adaptive Timeout Mechanism based on Resource Preference SDN Flow Entry Adaptive Timeout Mechanism based on Resource Preference. *IOP Conf. Ser. Mater. Sci. Eng.* **2019**, *569*, 042018. [CrossRef]
13. Panda, A.; Samal, S.S.; Turuk, A.K.; Panda, A.; Venkatesh, V.C. Dynamic Hard Timeout based Flow Table Management in Openflow enabled SDN. In Proceedings of the 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN), Vellore, India, 30–31 March 2019; pp. 1–6.
14. Kim, E. Enhanced Flow Table Management Scheme with an LRU-Based Caching Algorithm for SDN. *IEEE Access* **2017**, *5*, 25555–25564. [CrossRef]
15. Khan, M.K.A.; Sah, V.K.; Mudgal, P.; Hegde, S. Minimizing Latency due to Flow Table Overflow by Early Eviction of Flow Entries in SDN. In Proceedings of the 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Bangalore, India, 10–12 July 2018; pp. 1–4.
16. Zarek, A. *OpenFlow Timeouts Demystified*; University of Toronto: Toronto, ON, Canada, 2012.
17. Li, Q.; Huang, N.; Member, S.; Wang, D.; Li, X. HQTimer: A Hybrid Q-Learning-Based Timeout Mechanism in Software-Defined Networks. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 153–166. [CrossRef]
18. Li, Q.; Liu, Y.; Zhu, Z.; Li, H.; Jiang, Y. BOND: Flexible failure recovery in software defined networks. *Comput. Netw.* **2019**, *149*, 1–12. [CrossRef]

19. Liu, Y.; Tang, B.; Yuan, D.; Ran, J.; Hu, H. A Dynamic Adaptive Timeout Approach for SDN Switch. In Proceedings of the 2016 2nd IEEE International Conference on Computer and Communications (ICCC), Chengdu, China, 14–17 October 2016; pp. 2577–2582.
20. Nguyen, X.; Saucez, D.; Barakat, C.; Turletti, T. Rules Placement Problem in OpenFlow Networks: A Survey. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 1273–1286. [CrossRef]
21. Ma, Z. The Research on Flow Table Optimization Based on Dynamic Timeout Mechanism. In Proceedings of the 2019 IEEE International Conference on Power Data Science (ICPDS), Taizhou, China, 22–24 November 2019; pp. 8–12.
22. Sooden, B. A Dynamic Hybrid Timeout Method to Secure Flow Tables Against DDoS Attacks in SDN. In Proceedings of the 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC), Jalandhar, India, 15–17 December 2018; pp. 29–34.
23. Science, C. A Hybrid-timeout Mechanism to Handle Rule Dependencies in Software Defined Networks. In Proceedings of the 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Atlanta, GA, USA, 1–4 May 2017; pp. 241–246.
24. Yang, H.; Riley, G.F. Machine Learning based Flow Entry Eviction for OpenFlow Switches. In Proceedings of the 2018 27th International Conference on Computer Communication and Networks (ICCCN), Hangzhou, China, 30 July–2 August 2018; pp. 1–8.
25. Guo, Z.; Liu, R.; Xu, Y.; Gushchin, A.; Walid, A.; Chao, H.J. STAR: Preventing flow-table overflow in software-defined networks. *Comput. Netw.* **2017**, *125*, 15–25. [CrossRef]
26. Yang, H.; Riley, G.F.; Blough, D.M. STEREOS: Smart Table EntRy Eviction for OpenFlow Switches. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 377–388. [CrossRef]
27. Kim, N.; Kim, D.; Jang, Y.; Lee, C.; Lee, B. Applied sciences Traffic Characteristics in Software-Defined Networks. *Appl. Sci.* **2020**, *10*, 3590. [CrossRef]
28. Li, H.; Guo, S.; Wu, C.; Li, J. FDRC: Flow-Driven Rule Caching Optimization in Software Defined Networking. In Proceedings of the 2015 IEEE International Conference on Communications (ICC), London, UK, 8–12 June 2015; pp. 1–6.
29. Benson, T.; Akella, A.; Maltz, D.A. Network Traffic Characteristics of Data Centers in the Wild. In Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, Melbourne, Australia, 1–3 November 2010.
30. Shirali-shahreza, S.; Ganjali, Y.; Member, S. Delayed Installation and Expedited Eviction: An Alternative Approach to Reduce Flow Table Occupancy in SDN Switches. *IEEE/ACM Trans. Netw.* **2018**, *26*, 1547–1561. [CrossRef]