Received October 13, 2020, accepted November 3, 2020, date of publication November 6, 2020, date of current version December 1, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3036437

Deploying a Fast Detection and Eviction Mechanism of Invalid Connection-Oriented Flow-Entries in SDNs: A Scalability Approach

WEN-KANG JIA, (Senior Member, IEEE), RUOLAN YING, AND XIAONING SHI
College of Photonic and Electronic Engineering, Fujian Normal University, Fujian 350007, China

Corresponding author: Wen-Kang Jia (wkjia@fjnu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61871131 and Grant U1805262.

ABSTRACT As a new paradigm of network architecture, Software Defined Networking (SDN) has been used in a large number of scenarios because it realizes flexible and efficient fine-grained flow control of the network, and promotes the evolution of the network to a programmable and scalable direction. However, the transition of the traditional networking model to SDN architectures poses scalability issues due to the limitation of the flow-table in size. Facing the traffic explosion on future networks with resource-constrained architectures, the storage space of the flow-table is not enough to bear so many flow-entries so that it not only causes performance degradation in data delivery but also results in scalability and cost-efficiency issues. To address this issue, in this article, we propose a solution to expedited evict the invalid flow-entries by detecting the disconnect messages of connection-oriented protocols such as Transmission Control Protocol (TCP) and Stream Control Transmission Protocol (SCTP) based on SDN controller and OpenFlow or programming protocol-independent packet processors (P4) switches. The behavior of detection is achieved by adding a specific SDN ruleset within the transport-layer in between the controller and switches. Different from the original timeout solutions, our scheme can delete invalid flow-entries in time according to the transmission layer disconnection instead of relying on the original timeout mechanism. Through a series of simulation results. we also demonstrate the superiority of our proposed solution in reducing the flow-entries occupancy and control overhead on controller, and improving the table-miss rate.

INDEX TERMS SDN, TCP, SCTP, P4, flow-table, eviction, scalability.

I. INTRODUCTION

At present, with the explosive growth of various networks and applications, networks and services are becoming more and more complex and diverse than ever before. In the context of such network environments, how to quickly and dynamically build networks, rapidly deploying services, and efficiently establishing connections are still critical to the future [1]. In order to solve the problems faced by the current traditional network, around the world researches have begun on the next generation of the Internet, such as GENI (Global Environment for Network Innovations) [2] in the United States, FIRE (Future Internet Research and Experimentation) [3] in the European Union and so on. Based on the preliminary works that have been carried out, ForCES (Forwarding and control element separation) [4], 4D architecture [5], RCP (Routing Control Platform) [6], SANE (Secure Architecture for the Networked Enterprise) [7], and Ethane [8] are the most notable examples that share the details of implementation and measured performance of concrete network designs.

As a new paradigm of network architecture, SDN (Software Defined Networking) enhances the flexibility and openness of the network. In 2008, OpenFlow [9] was proposed and gradually extended it to the SDN. Its basic attributes include the concept of control-data split, logically centralized control of networks, and flexible open interface to program underlying network infrastructure, where the northbound interface implements efficient bearer for services, and the southbound interface controls the forwarding and packet processing of network devices through the open standard protocols such as OpenFlow.

One of the advantages of SDN being able to develop so quickly is that SDN allows fine-tuned control of flows and achieves managing every flow individually. New controller applications usually take advantage of this and proactively

The associate editor coordinating the review of this manuscript and approving it for publication was Abderrahmane Lakas.

install rules for new flows [9], [10]. However, this level of granularity will cause the problem of SDN scalability due to the large scales and high load network traffic. This is because the SDN switches widely use ternary content addressable memory (TCAM) to store their forwarding rules given by an SDN controller currently. Practically, the size of TCAM-based flow-table in an SDN switch is restricted to a few hundred or at most thousand [11]–[13] of entries due to manufacturing cost and high-power consumption. The limited size of the TCAM-based flow-table is not sufficient to manage the massive data flows in a large-scale network, which may result in the flow-table overflow problem [9], [11], and decreases the feasibility to implement a large-scale SDN [14], [15]. But the SDN controller might need to install at least one flow-entry per flow in each switch along the end-to-end path used by the flow to realize per-flow based fine-grained traffic control. This mechanism will consume the available TCAM space on SDN switches quickly under heavy traffic load conditions [15], [16]. Therefore, as the network scale increases, the available TCAM space will be consumed quickly by all the flow-entries generated during the operation of the SDN network, which will further restrict the network scale and application diversity. Generally, the invalid flow-entries are removed from flow tables in two ways, either at the request of the controller or via the flow expiry mechanism of the switch. Practically, the flow expiry mechanisms (e.g., hard-timeout or idle-timeout) play a major role in the regulation of flow-table space, without major relying on the controller. However, to choose a suitable timeout value is a complicated work: long timeouts will increase the storage burden for the TCAM space on SDN switches due to flow-table occupancy. On the contrary, small timeouts might result in unnecessary control signaling traffics [9], [17] just like that removed by the controller approach.

In this article, we have abandoned the approach relying on the timeout to delete expired flow-entries, and propose an expedited invalid flow detection and eviction scheme that focuses on connection-oriented protocols such as Transmission Control Protocol (TCP) and Stream Control Transmission Protocol (SCTP) in the typical IP-based SDN framework today. Our solution is based on the following technique: we expedite the eviction of invalid flow-entries by detecting the termination from connection-oriented protocols. We rely on detecting the transport-layer disconnect messages such as the TCP-FIN and SCTP-SHUTDOWN-COMPLETE to instruct that the TCP and SCTP flows are terminated respectively. We compare our expedited eviction solution with current OpenFlow ordinary operations (i.e., hard timeout and idle timeout) on two different scenarios. The series of simulation results show that this simple change has a major impact on reducing the flow-table occupancy and significantly improves the scalability of SDN for future requests.

The remainder of the paper is organized into the following sections. In Section II, the background and the related works are described. In Section III, we discuss the mechanism of our proposed scheme. In Section IV, we present the simulation environment and evaluate the impact of our proposed scheme and ordinary operations on the flow-table occupancy in different scenarios. Finally, Section V summarizes our work.

II. BACKGROUND AND PREVIOUS WORKS

This section begins with a brief introduction to the mechanism of OpenFlow timeout and then introduces the previous works and researches which focus on addressing the scalability of SDNs.

A. THE FLOW EXPIRY MECHANISMS AND THEIR DISADVANTAGES

The flow-entries are usually removed from flow-tables in two ways in Openflow-based SDNs, either at the request of the controller (centralize approach) or via the switch flow expiry mechanism (decentralized approach). The switch flow expiry mechanism is running by the switch independently of the controller and is based on the state and configuration of flow-entries [9], which is mainly associated with the hard_timeout and idle_timeout existing in each flow-entry. For the hard_timeout, the non-zero hard_timeout field will cause the specific flow-entry to be evicted after the flow-entry has exceeded the pre-configured survival time, no matter whether the flow-entry is needed to match the follow-up traffic later on. Hence, the hard_timeout is the absolute time that the flow-entry is removed from the switch. For the idle_timeout, the switch must record the last matched packet's arrive time which is associated with the flow-entry when the idle_timeout field is a non-zero value, because the switch will actively remove the flow-entry from the flow-table, if no packet matches the flow-entry exceed the given idle_timeout period. That is, the idle_timeout is the relative time that the flow-entry is removed from the switch. If the two timeout fields are different, such as one is a zero field and another is a non-zero field, then the flow-entry will be removed depending on the timeout mode of the non-zero field. And when the field of both timeout fields are the non-zero values, the selection of the flow-entry deletion mode mainly depends on the values of the hard_timeout and the idle_timeout. If idle_timeout is less than hard_timeout, the switch deletes the flow-entry when no packet matches the flow-entry during the idle_timeout period; the switch still evicts the flow-entry when the flow-entry exceeds the hard_timeout, even if a packet has matched the flow-entry during the idle_timeout period. In contrast, if idle_timeout is greater than hard_timeout, the switch evicts the flow-entry from the flow table when the hard_timeout period has arrived before the idle_timeout period, and optionally notifies the controller the flow was removed [18].

From the above description, we can clearly aware that the hard_timeout is not flexible enough, which cannot be efficient for dealing with dynamically changing traffic patterns. For example, if a frequently matched flow-entry is removed due to a hard_timeout, then the packet that should match this flow-entry will trigger the `Packet-In` event, increasing the burden on the controller. Therefore, the controller typically sets an idle_timeout for the flow-entry, not a hard_timeout. However, there is a tradeoff between a long idle_timeout and a short idle_timeout, and different SDN networks have different requirements for the value of idle_timeout, so it is not optimal to rely on idle_timeout to delete invalid flow-entries. Hitherto, various values of hard_timeout have been employed in SDN switches, reasonable ranging from 5 seconds [19] to 60 seconds [20].

B. PREVIOUS WORKS

The research on flow expiry and eviction mechanisms for SDN have been inspired by the limited size of the flow-table and increased burden of the controller with incremental stale flow-entries. Since there are several limitations for flow expiry mechanisms to detect flow completion precisely, the available flow-table space is occupied by a certain amount of invalid flow-entries before they are evicted. As a consequence, SDN decreases the level of scalability by providing an ineffective way of maintaining the invalid flow-entries in accordance with the SDN specification. The scalability of the SDN switch thus has been recognized as the barrier of SDN technology, currently, there is plenty of existing works have been proposed to improve the effective management of the flow-tables by increasing the flow-table space utilization.

In [21], the DIFANE architecture is proposed, which divides the SDN switches into two categories: ordinary switches and authority switches. All the SDN network flow-entries are stored in authority switches uniformly, and the new flow-entries are stored in ordinary switches. When the new flow-entries fail to match in the ordinary switches, the ordinary SDN switch sends a request to the authoritative SDN switch obtaining the corresponding flow-entries to complete the forwarding work.

Similar to DIFANE, the authors in [22] addressed the scalability of SDN by adding SDN software switch. The SDN software switch is used to process new flows that cannot be processed by the ordinary switch and store most of the flow-entries. However, during the operation of the SDN network, network traffic is dynamically changed, so these two schemes will bring great challenges to achieve storing all flow-entries in switches.

A dynamic timeout control algorithm is proposed in [23], which the controller collects various traffic parameters from the switches and predicts the inter-arrival times of packets in a flow, the idle_timeout value of each flow is thus dynamically adjusted by the controller according to the traffic information of the switches without modifying the switch configuration.

The authors in [24] proposed a scheme that uses a speculative mechanism to predict whether a flow entry has expired, and thus can remove the corresponding flow entry from the switch in advance.

Based on a similar ideal, SmartTime was proposed in [25] based-on an adaptive timeout heuristic algorithm to calculate the most appropriate idle_timeout value individually for each practical flow instead of setting the same timeout value to each flow at the switch. This scheme sets and adjusts the value of idle timeout by observing the interval of the arrival of the adjacent packet to the switch, resulting in reasonable utilization of the TCAM. At the same time, when the flow-table is about to overflow, the scheme randomly deletes the flow-entries in the flow-table to increase flow-table space usage. However, there is a great risk of causing the flow interruption to occur since the flow-entry is randomly removed, increasing the load of the SDN controller and reducing the performance of the SDN network.

A flow table management method is proposed in [26], which the caching algorithm based on least-recently-used approach is employed to keep flow entries in SDN switches as long as possible. In the proposed scheme, an invalid flow entry is evicted when its age reaches a certain threshold, rather than when its idle_timeout expires. The switch adjusts its cache size according to the vacancy of flow table and the controller thus determines the packet forwarding path through the switches by referring to the vacancies.

The author in [27] propose a SofTware-defined Adaptive Routing (STAR), to mitigate switch flow table overloading issue, which SDN controller collects the real-time flow-table utilization from each switch, intelligently evicts invalid flow-entries when needed to accommodate new flows, and selects routing paths for new flows based on flow-table utilizations of each switch across the network, thus maximizing network scalability and throughput.

The author in [28] also propose a dynamic routing approach named DIFF that differentiate flows based on their impact on network resource and adaptively selects appropriate routing paths for them to balance the utilization of flow tables globally, and to mitigate the problems of flow-table overflow and inefficient bandwidth allocation, while increasing network throughput.

In [29], the authors propose a dynamic way to allocate an adaptive hard_timeout value for each flow entry which reflects the distinct flow characteristics including predictable and unpredictable flows. Compared to statically allotted hard timeout, the dynamically allotted hard timeout approach performs better performances.

The Tag-In-Tag [30] scheme is proposed by S. Banerjee *et al.*, which uses the path tag and the flow tag to identify the flow forwarding path and flow-entry in the network. Compared to flow-entries, Path Tag and Flow Tag use fewer bit bytes, occupying less flow-table space. However, there is a defect in this solution, which modifies the internal physical structure of the SDN network switch, and it is not conducive to promote in the actual networks.

In [31], the authors propose an SDN-based TCP congestion control mechanism—SDTCP, which can accurately decelerate the rate of background flows by measuring the TCP traffics through detecting the TCP flags such as SYN, ACK, and FIN via SDN/Openflow technologies.

The concept in [32] is follow a similar approach for solving the problem of SDN scalability, which presents a hybrid flow rule placement approach, where the flow hit ratio and flow table occupancy are used for flow classification. Based on the classification, two schemes focusing on TCP flows and non-TCP flows are proposed, respectively. For TCP flows, they consider FIN and RST flags to be the last packet and remove the related flow-entries after that. For non-TCP flows, this approach comes at the cost of missing the first few non-TCP flows and installs the related flow-entries only when it exceeds a predetermined number of packets. However, the authors of the paper did not provide a detailed description of how to implement this mechanism, just simply introduce an idea. In contrast to it, in our proposed scheme, we not only detail the implementation principle but also evaluates the comprehensive performance of the proposed scheme such as the maximum number of flows per network and table-miss rate.

Following a similar approach again, the article [33] proposes a flow entry removal scheme based on programming protocol-independent packet processors (P4) technology. After the P4 switch parses the TCP header and checks the FIN or RST flags. The P4 switch thus can determine whether a TCP connection has closed, and remove the corresponding invalid flow entry and send a notify message to the controller.

III. PROPOSED SCHEME

It is well known in the networking industry that the connection-oriented transport-layer protocol needs to establish a connection when establishing a communication session and needs to disconnect when communication is terminated. Our proposed scheme here is to use a connection-oriented transport-layer protocol (layer-4) features to detect these events, and the framework should be used in conjunction with the standard SDN environment. Flag fields are commonly used in various transport-layer protocol headers to specify certain actions, and can be used to detect the start and the end of connections. In the proposed scheme, we rely on control signaling of TCP (referred to as flag) and SCTP (referred to as chunk type) to detect imminent flow completions, the corresponding invalid flow-entries no longer depends on a flow expiry mechanism but on each of flow-entry can count an Active Connection Counter (ACC), hence evict the flow-entry immediately once the countdown has been reached zero. The proposed scheme is based on previous works in [34] and [35] of the authors, which approaches have also extended to take into account the integration of both the TCP and SCTP connections are exist in an SDN. In addition, more comprehensive and rigorous performance evaluations for the proposed scheme will be conducted in the following sections.

A. EXPEDITED INVALID TCP FLOW DETECTION AND EVICTION SCHEME

In this subsection, we introduce the expedited invalid TCP flow eviction scheme to achieve the goal of evicting the invalid flow-entries upon the flow-table immediately, which is by adding a specific SDN ruleset named "TCP flag detector" upon switches for monitoring and counting all valid TCP connections, and establishing and evicting the corresponding forwarding rules by the specific SDN application upon controller. The key concept of the TCP flag detector is to discover the TCP control messages such as the SYN (synchronization), ACK (acknowledgment), RST(reset), and FIN (final) flags between the client and server timely. Therefore, once the last TCP connection has been terminated, it implies that the corresponding flow-entry is invalided, thus the SDN controller can quickly notified to evict the practical invalid flow-entries which store in the SDN switches.

As we all know, multiple TCP socket connections can be established between both communication parties simultaneously. On the other hand, multiple flows with identical destination or prefix might match a wildcard rule in a practical SDN environment. Hence, we setup an Active Connection Counter (ACC) for each flow-entry in the proposed scheme to record the active number of concurrent TCP sessions. The detail match fields and related activities of Rule 0 and Rule 2/3 are shown in Table 1 and Table 2. As Table 1 shows, the main information of the match fields includes TCP-SYN, TCP-FIN, TCP-RST, and TCP-ACK in Rule 0, and the corresponding triggering behavior will issue the information of `Packet-In` to the controller. As with the mechanism of Rule 0, the corresponding match fields will trigger the action of forwarding in Rule 2/3. And note that Rule 0 is for all TCP control flows pass through the switch, while Rule 2/3 is for specific TCP flows with the identical destinations or prefixes.

TABLE 1. The Match Fields and Related Actions of TCP Flag Detector (Rule 0).

Match fields	Action
(`IP PROTO = TCP` and `IPV4 SRC: IPV4 DST = *:*`) and (`TCP FLAGS = TCP-SYN` or `TCP FLAGS = TCP-ACK` or `TCP FLAGS = TCP-RST` or `TCP FLAGS = TCP-FIN`)	`Packet-In`

TABLE 2. The Match Fields and Related Actions of Forwarding Rule 2/3.

Match field	Action
`IPV4 DST = 203.100.1.*`	`Forward to Port X`
`IPV4 DST = 140.23.5.*`	`Forward to Port Y`

In order to better describe the mechanism, the example of the proposed scheme is shown in Figure 1, where the left-half of the figure is a Finite State Machine (FSM) in the perspective of SDN controller, and the right-half of the figure is a message flow chart for proposed scheme respectively. It can be seen that the SDN controller will first pre-issue a `Flow-Mod` message toward all subordinate switches to instruct them to add the Rule 0 of the specified TCP flag detector before establishing any end-to-end TCP connection. In TCP standard, a well-known three-way handshake is requiring both the TCP client and server to exchange SYN and ACK messages before actual data delivery begins. During the TCP connection establishment stage, once the

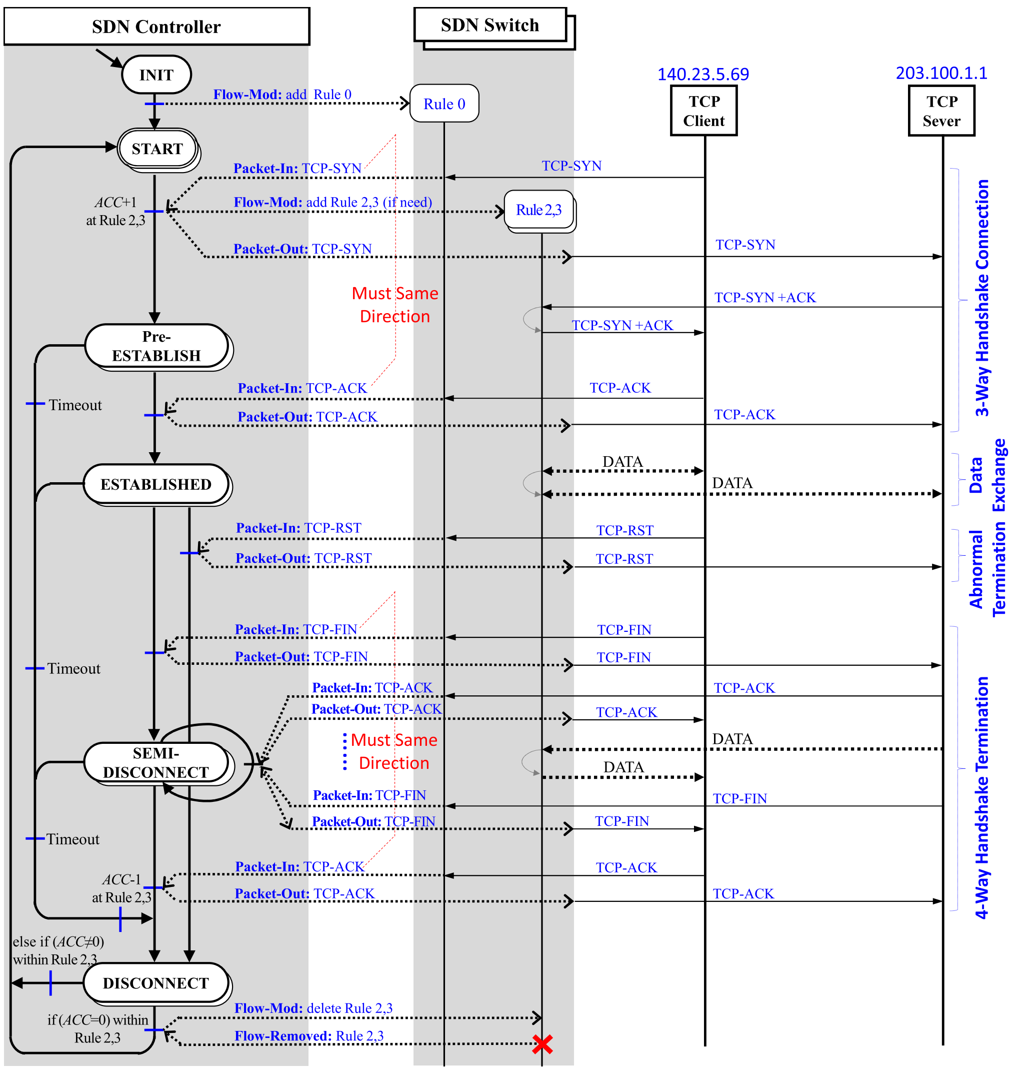


FIGURE 1. Finite state machine and protocol sequence diagram of expedited invalid TCP flow detection and eviction scheme.

TCP client sends a SYN message to the TCP server, it means that a new TCP connection to be established between two sides. In the proposed approach, the TCP-SYN message will first be captured by Rule 0 and then formed a `Packet-In` message including the whole TCP control packet, which is conveyed by matching Rule 0 and then transmitted back to the controller. Subsequently, the controller generates the corresponding forwarding Rule 2 and Rule 3, and deploys they toward each switch along the forwarding path through the `Flow-Mod` message, and forwards the pending SYN message through `Packet-Out` messages to resume the TCP association procedure. From the FSM's perspective, the automata would transition from state "START" enter to state "Pre-ESTABLISH" now, whereas the ACC increased. Afterward, when a ACK message on the TCP connection parameters is captured in same direction as expected, through `Packet-In` and `Packet-Out` procedures, indicating that the connection establishment is successful and resources are committed on both TCP sides. Additionally, with regard to related rules, named Rule 0 and Rule 2/3, the related information of the incoming and outgoing bitstreams between the client and the server will be uniquely hit according to the match fields either Rule 0 (specified TCP commands such as TCP-SYN) or Rule 2/3 (ordinary forwarding), and perform corresponding operations according to the action of Rule 0 and Rule 2/3. As for Rule 2 and 3, they are both ordinary forwarding rules, where Rule 2 is a forward rule from the client to the server, and Rule 3 is the backward rule of Rule 2. Note that the Rule 0 does not attend to the TCP-SYN+ACK, which can be treated by the forwarding Rule 2/3 and will ordinary forwarded to the final destinations. From the FSM's perspective, the automata would transition from state "Pre-ESTABLISH" enter to state "ESTABLISHED", and entering the normal TCP data exchange phase now.

During the TCP data exchange stage, once an unexpected TCP packet arrives at an end-host, that the end-host usually responds by sending an RST message back on the other side and closes such connection immediately. If an end-host receives a RST message in either side, it assumes an error has occurred and also closes the connection immediately under TCP regulation. In such abnormal condition, Rule 0 will catch and `Packet-In` and `Packet-Out` such TCP-RST message. From the FSM's perspective, the automata would transition from the "ESTABLISH" state enter into the "DISCONNECT" state directly, whereas countdowns the ACC. Due to the TCP connection is considered to be half-closed when it's closed in one direction and still open in the other direction. Therefore, the TCP needs to fully terminate the connection by a 4-way handshake and to allow incomplete data transfer after receiving a first FIN message from other side. As analyzed in [32]: there are more than 97% of TCP flows arriving at most 2 packets after receiving FIN message, and a FIN message is the last packet of TCP flows in half of these cases. In order to support the half-closed mechanism, there are special provisions in determining whether the TCP connection is really terminated. When one of TCP party (such as a client) initiates a disconnect request, the FIN flag of TCP message is captured by the Rule 0 and notified to the controller via a `Packet-In` message, and uses the `Packet-Out` message to begin the disconnect process. After receiving a TCP-FIN message from any communication party (such as a client side in the instance), TCP will be entering into a half-closed state, during which the client might continue to accept unfinished data stream from the server. Until catching the TCP-FIN message from the other direction (server side), the half-closed state will really terminate. For terminating the TCP connection, we consider that only detecting the TCP-ACK messages from the identical direction (such as TCP-ACK messages from the client with an identical socket of previous TCP-FIN) can be ensured the TCP disconnected completely, and the TCP-ACK messages from distinct sockets cannot be triggered. Therefore, when all active TCP connections are terminated by identified through ACC $= 0$, the flow-entries of Rule 2 and 3, that flow-entry are assumed invalid and can be evicted as soon as possible. And the released flow-entries will be available for the newly arrived flow, to achieve reducing the occupancy of the flow-table and improving the scalability of the SDN.

From the FSM's perspective on the SDN controller, the controller should maintain an ACC and associated function for each valid flow-entry. The ACC function is a recursive function that can repeat itself several times, outputting the concurrent number of active TCP connections and the end of each iteration. In the proposed scheme, the automata would first transition from "START" state enter into the "Pre-ESTABLISH" state correspond to opening a connection by a TCP client issues a TCP-SYN message, and whereas the ACC $\leftarrow$ACC+1 is performed. Once a TCP-SYN is captured in the same direction again, it enters into the "ESTABLISHED" state. Otherwise, an abnormal incomplete TCP association will cause the timeout and ACC$\leftarrow$ACC-1, whereas it enters back to the "START" state. Then the FSM will accomplish the 3-way handshake process to enter the TCP data transmission stage. When the controller receives a TCP-FIN flag in this state it will enter the "SEMI-DISCONNECT" state, after a TCP-SYN is captured in the same direction again, it enters into the "DISCONNECT" state. This happens whenever either TCP connection side starts a disconnect process that is expected to wait for incoming TCP-FIN requests. At this moment, the ACC$\leftarrow$ACC-1 is also performed, and once an ACC value that corresponding to specific flow-entry is count-down to zero, it implies the flow-entry is no longer valid for any active TCP connection flow, the controller should send a `Flow-Mod` message with the delete the command to the switch to remove the invalid flow-entry immediately. Then the FSM will accomplish the 4-way discount process to move back to the "START" state.

In order to avoid the TCP connection terminating abnormally, for example, the FIN and ACK messages are eventually not appearing for captured, the proposed scheme should be used in conjunction with the original idle timeout method in SDN. Once the forwarding rules 2 and 3 fails to match any TCP data packet as expected, the idle timeout mechanism will start the flow-table eviction. Besides, considering that an SDN wildcard forwarding rule could be designed to provide multiple flows at the same time. Therefore, the SDN controller must compare and confirm that the so-call invalid flow-entry is indeed useless for any other active flows e.g., non-TCP flows before performing the flow-table eviction action.

B. EXPEDITED INVALID SCTP FLOW DETECTION AND EVICTION SCHEME

Similar to TCP, the SCTP is also a connection-oriented IP transport protocol in nature, which can provide reliable transmission and perfect congestion control, but the SCTP is a broader concept than the TCP [36]. For example, SCTP can protect the endpoints from a Denial-of-Service (DoS) attack through the four-way handshake: INIT, INIT_ACK, COOKIE_ECHO, and COOKIE_ACK and provides the multi-streaming and multi-homing functionality to extend the availability of the application. For disconnection, there is 3-way handshake: SHUTDOWN, SHUTDOWN_ACK and SHUTDOWN_COMPLETE and according to [36], [37], during the shutdown, the half-open state (like TCP) is not be supported wherein any endpoint performs a shutdown, the new data from its user will be rejected. SCTP also supports the heartbeat mechanism for monitoring the state of the system and coordinating the work of the master and slave servers. It can detect faults in the server application-level system, isolate and recover the error quickly, maintain system availability, and ensure that critical services continue to be highly available.

Because of the superiority of SCTP, an increasing number of recent applications have incorporated their own applications on top of SCTP instead of TCP. Therefore, we choose the SCTP to combine with our concept, using the SCTP detection mechanism to achieve real-time monitoring, to minimize occupancy in a flow-table and delete

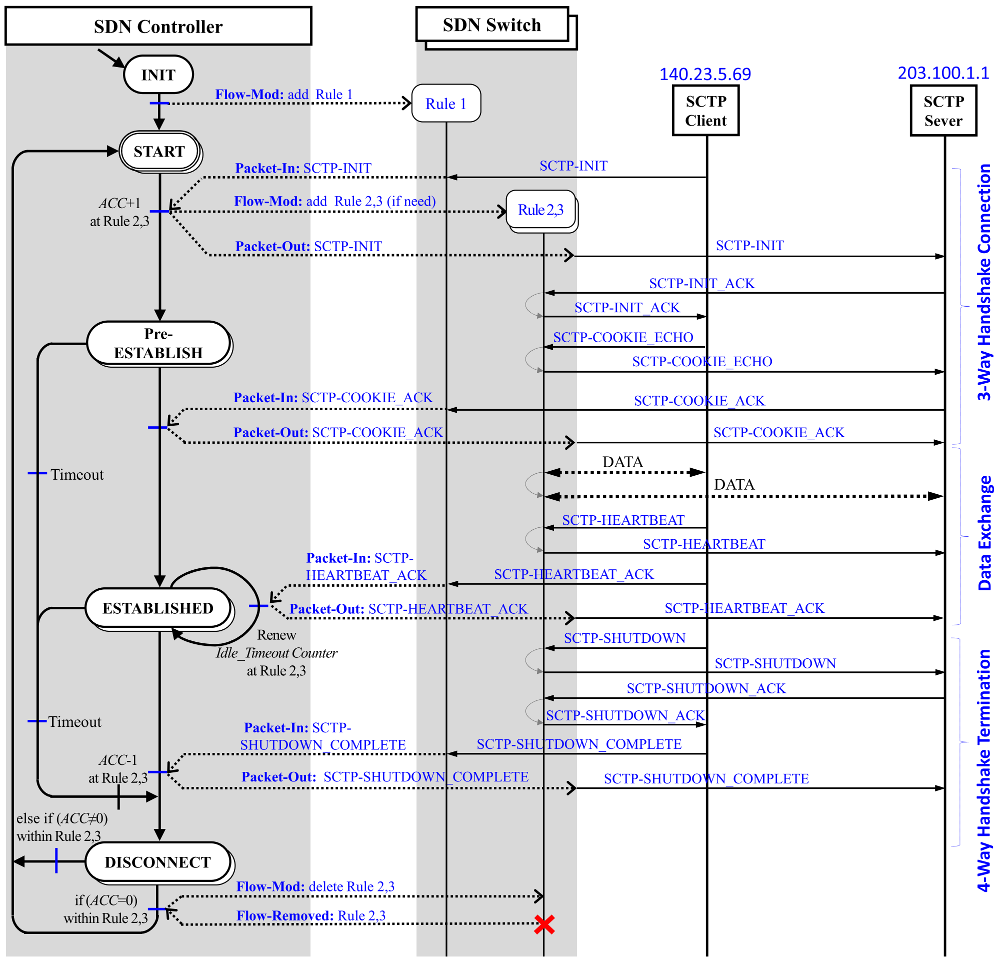


FIGURE 2. Finite state machine and protocol sequence diagram of expedited invalid SCTP flow detection and eviction scheme.

expired flow-entries in an SDN switch as soon as possible. We propose a flow-table management scheme—the expedited invalid SCTP flow eviction scheme, which is by adding a specific SDN rule set named "SCTP chunk-type detector" upon switches for monitoring and counting all valid SCTP connections, and establishing and evicting the corresponding forwarding rules by the specific SDN application upon controller. In order to achieve the goal of expedited eviction the invalid flow-entries in the flow-tables, the key concept of the SCTP chunk-type detector is to discover the SCTP control messages such as the SCTP-INIT, SCTP-COOKIE_ACK, SCTP-HEARTBEAT_ACK, and SCTP-SHUTDOWN_COMPLETE between SCTP endpoints, through real-time formulating related rules about its match fields and action, thus quickly notifies the SDN controller to delete the invalid flow-entries which store in the SDN switches. In addition, in terms of related rules, which mainly includes three rules, named Rule 1 and Rule 2/3. By setting the specified match fields and the action, we make the incoming and outgoing bitstreams can only match the unique rule and execute the related action. As for Rule 2 and 3, they are both ordinary forwarding rules, where Rule 2 is a forward rule from the client to the server, and Rule 3 is the backward rule of Rule 2. Additionally, note that the Rule 1 does not attend to the SCTP-INIT_ACK, SCTP-COOKIE_ECHO, SCTP-HEARTBEAT, SCTP-SHUTDOWN, and SCTP-SHUTDOWN_ACK which can be treated by the forwarding Rule 2/3. An identical ACC for each flow-entry in the proposed scheme to record the active number of concurrent SCTP sessions. Any changes made to the ACC will automatically available to both TCP and SCTP connections, as well as to any other future traffic flow are sharing the same entry with.

The example of the framework operation of our proposed scheme is shown in Figure 2, where the left-half of the figure is a FSM in the perspective of SDN controller, and the right-half of the figure is a message flow chart for proposed scheme respectively. The SCTP chunk-type detector will placed on the SDN switches named Rule 1, where the match field in rules ensure the relevant control-bit field between the SCTP client and server match the corresponding rules and only match and the action ensure the corresponding operation is performed. Note that older OpenFlow such as version 1.0 [38] does not directly support matching on SCTP header flags, but it can be extended by extensible matching supported in newer OpenFlow specification [9].

Similar to the operations of expedited invalid TCP flow eviction scheme. The SDN controller will first pre-issue a Flow-Mod message toward all subordinate switches to instruct them to add the Rule 1 before establishing any end-to-end SCTP connection. Consider that SCTP uses a 4-way handshake between client and server to enable secure connections. At the beginning of establishing an end-to-end SCTP connection, the negotiation will take place in two rounds. In the first-round, upon receipt of the `Packet-In` message with SCTP-INIT from SCTP client to server, which the message is conveyed by matching the Rule 1, the controller generates a forwarding rules (Rule 2/3) and deploys it toward each switch along the forwarding path, and forwards the SCTP-INIT through `Packet-Out` messages to realize the SCTP association procedure. In the second round, an SCTP-COOKIE_ACK message on the SCTP connection parameters is returned, indicating that the connection establishment was successful and resources committed on both sides.

During the SCTP association phase, the related information between the endpoints will be uniquely hit according to the match fields either Rule 1 (specified SCTP commands such as SCTP-COOKIE_ACK) or Rule 2/3 (ordinary forwarding), and perform corresponding operations according to the action of Rule 1 and Rule 2/3. Thus, the Rule 1 will catch and `Packet-In` the SCTP-COOKIE_ACK message and the Rule 2/3 will ordinary forward the SCTP-INIT-ACK and SCTP-COOKIE-ECHO messages to the final destinations. The detail match fields and related activities of Rule 1 is shown in Table 3, the main information of the match fields includes SCTP-INIT, SCTP-COOKIE_ACK, SCTP-HEARTBEAT_ACK, and SCTP-SHUTDOWN_COMPLETE in Rule 1, and the corresponding triggering behavior will issue the information of `Packet-In` to the controller. Same as the mechanism of Rule 1, the corresponding match fields will trigger the action of forwarding in Rule 2 as in Table 2. From the FSM's perspective, the automata would transition from state "Pre-ESTABLISH" enter to state "ESTABLISHED", and entering the normal SCTP data exchange phase now. Before the SCTP association is really established, if Rule 1 cannot detect the SCTP-COOKIE-ACK within the prescribed time limit, indicating that the actual SCTP connection between client and server has not established successfully, which the SCTP-INIT message considers being a DoS attack. In the meantime, in order to suppress such a potential DoS attack, the controller could delete the corresponding flow-entries (Rule 2/3) stored in switches according to the flow expiry mechanism if ACC = 0.

TABLE 3. **The Match Fields and Related Actions of SCTP Flag Detector (Rule 1).**

Match fields	Action
(`IP_PROTO= SCTP` and `IPV4_SRC: IPV4_DST =*:*`) and (`SCTP_FLAGS= SCTP-INIT` or `SCTP_FLAGS= SCTP-COOKIE-ACK` or `SCTP_FLAGS= SCTP-HEARTBEAT-ACK` or `SCTP_FLAGS= SCTP-SHUTPDOWN-COMPLETE`)	`Packet-In`

During the SCTP data exchange stage, we consider that the shutdown of the SCTP connection means that the flow-entries stored in switches might also invalid. Therefore, when the Rule 1 monitors the process of the shutdown, especially detects the SCTP-SHUTDOWN-COMPLETE message, the SCTP chunk-type detector will inform the controller to evict the corresponding invalid flow-entries in switches as soon as possible once the last SCTP connection has been terminated (ACC = 0), thus significantly improving the scalability of SDN for future requests.

In contrast, in order to avoid erroneous eviction that may cause control signaling overhead problems. Considering that an active SCTP connection might keep silence for a contain time until exceed the idle-timeout, the SCTP detector (Rule 1) also monitors the process of heartbeat by detects the message of SCTP-HEARBEAT-ACK, the SCTP detector will inform the controller to renew the idle_timeout counter of corresponding flow-entries, thus significantly reducing the large signaling interaction between SDN controller and switches once an erroneous eviction is occurring. In addition, the idle_timeout mechanism is used as the last line of defense to delete SCTP flow-entries that have not been terminated by the normal SCTP handshake messages.

Obviously, the goal of SCTP is to provide a more rigorous framework for the design and implementation of long-term transport-layer IP data exchange manner, that inadvertently minimizes the complexity for SDN control signaling interactions for processing SCTP messages, compare to processing TCP messages.

IV. PERFORMANCE EVALUATION

A. SIMULATION ENVIRONMENT

To build a proof of concept implementation of the proposed scheme and demonstrate its performance, we construct a series of simulations to compare the performance of the proposed scheme against the SDN ordinary operations. The network simulator is based on self-developed C codes. According to previous researches, three approaches to SDN deployment can be considered—1) PS: proposed scheme, 2) IT: idle timeout scheme; and 3) HT: hard timeout scheme. We have designed several simulation conditions and their combinations to understand the performance variation in different scenarios. The simulations are evaluated in three distinct reference network topologies, i.e., 1) 52-node Fat-Tree; 2) 28-node Cost-239 European network, and 3) 14-node NSFNet represented by a non-weighted directed graph G(V, E). And the detail parameters in the simulation are indicated Table 4: we set up reasonable system parameters to pattern the SDNs and evaluate scenarios of network performances, where flow frequency λ connection per second (CPS) is used for representing the network traffic, and the

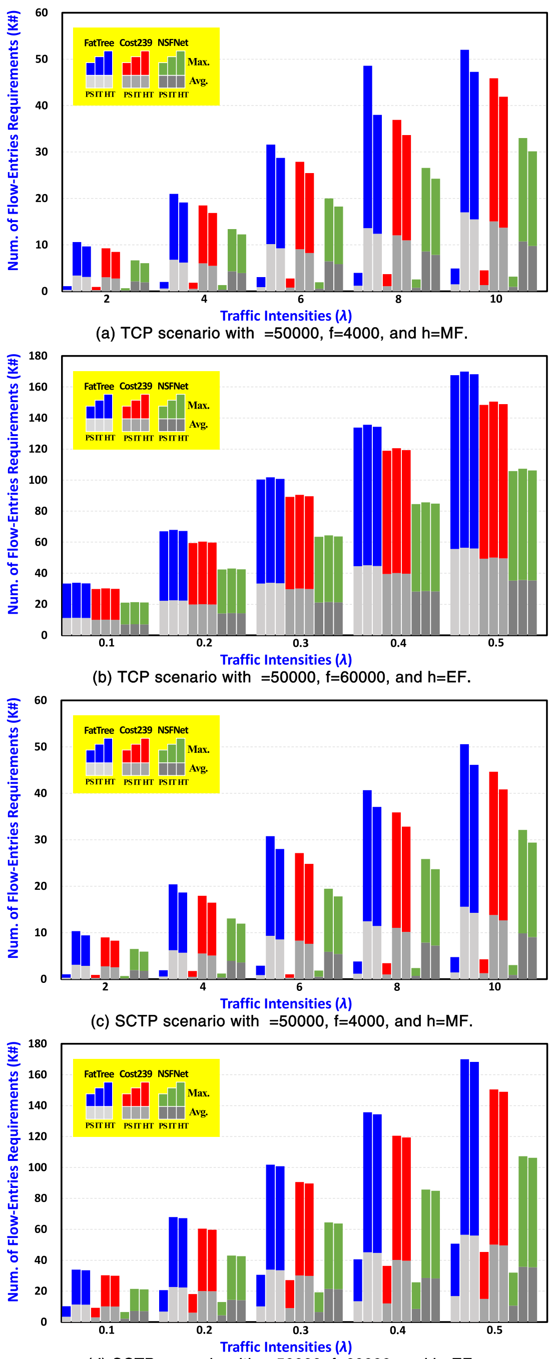
TABLE 4. System Parameters for Simulations.

System Parameters	Unit	Values
Reference network topology	G(V,E)	1) 52-node Fat-Tree 2) 28-node Cost-239 3) 14-node NSFNet
Flow arrival Rate (λ)	#/sec.	0.01∼50
Traffic type (h)	N/A	1) Mice-Flow (MF) 2) Elephant-Flow (EF)
Packet length distribution	byte	1) **MF:** avg. 64 bytes 2) **EF:** avg. 1000 bytes.
Flow duration	sec.	1) **MF:** avg. 30s; 2) **EF:** avg. 21600s.
Total flows (f)	#	1∼100,000
Link capacity	bit/sec.	∞
Flow-table size limitations (Φ)	#	1∼65,536
Flow-entry removal latency	sec.	1) Idle Timeout=60s; 2) Hard Timeout=60s.
Flow expiry mechanism	N/A	1) **PS:** Proposed Scheme 2) **IT:** Idle Timeout 3) **HT:** Hard Timeout

values of network traffic are set 0.01∼50, the total flows f is set the maximum to no more than 100,000, as for the flow-table limitation Φ will not exceed 65,535. We define the "mice-flow (MF)" as the flow with short holding time and packet length distribution in the simulation (average 30 seconds holding times) and relatively low traffic during the survival period, and the flow with relatively long survival time and long holding time in the simulation (average 21,600 seconds holding times) for "elephant-flow (EF)". According to the current reference, we set the timeout value to 60 seconds in either OpenFlow expiry mechanisms [20].

B. AVERAGE AND MAXIMUM FLOW-TABLE COMSUMPTIONS AT ENTIRE NETWORKS

As is known, the trend of the flow-table at the switch being occupied by the flow-entry during the whole simulation is first increased, then stabilized a period, and finally decreased towards 0. In order to describe the flow-table of all switches occupied by flow-entries during the whole simulation, we express it by the average flow-table occupancy rate and the maximum flow-table occupancy rate. In the simulation of the TCP scenario, Fig. 3(a) and 3(b) demonstrate the relationship between the number of flow-tables and different network traffic λ in constant flow-table limitations Φ under h = MF and EF conditions respectively. The lower part of the cylinder is expressed as the number of average flow-table occupancy, and the upper part is the number of maximum flow-table occupancy. We consider three distinct schemes: 1) PS represents the scheme we proposed (e.g., the first of the three cylinders under the same network traffic), 2) IT represents the method of idle_timeout (e.g., the second of the three cylinders under the same network traffic), and 3) HT represents the method of hard_timeout (e.g., the third of the three cylinders under the same network traffic). We can clearly see that, regardless of the type of network topologies, our method (especially under h = MF) not only performs well in the average number of flow-table occupancy but also has a significant optimization in the maximum number of flow-tables occupancy. As our proposed scheme can evict invalid flow-entries immediately during the entire simulation. It means that our proposed scheme makes the flow-table at all switches be occupied by fewer flow-entries averagely,



(a) TCP scenario with =50000, f=4000, and h=MF.

(b) TCP scenario with =50000, f=60000, and h=EF.

(c) SCTP scenario with =50000, f=4000, and h=MF.

(d) SCTP scenario with =50000, f=60000, and h=EF.

FIGURE 3. Number of flow-tables requirements vs. traffic intensities of three competed schemes under three reference topologies in four scenarios.

while the other two ordinary methods make the flow-tables occupancies relatively high during the whole simulation.

In the simulation of the SCTP scenario, Fig. 3(c) and 3(d) demonstrate the relationship between the number of flow-tables and different network traffic λ in constant flow-table limitations Φ under h = MF and EF conditions respectively. In order to represent certain malicious connections that have existed in the real world. We manipulated the situation that there are 90% of the SCTP connections have successfully established, while the remaining 10% indicates that the SCTP-INIT message has been sent but failed to establish, which might due to the SYN flooding attack. From the trend of Fig. 5 and 6, there are similar to Fig. 3 and 4. The method of expedited invalid SCTP flow eviction also performs excellently in reducing flow-table occupancy, not only under h = MF but also under h = EF.

C. FLOW-TABLE COMSUMPTIONS AT INDIVIDUAL SWITCHES

In the simulation of the TCP scenario, Fig. 4(a) and 4(b) present the probability density function (PDF) of the average flow-table consumptions for each individual switch with the h = MF and EF conditions respectively during the simulation for our proposed scheme and two ordinary flow expiry mechanisms, where PDF indicates the statistical number of switches which are occupied by the same scale of the number of flow-entries during the simulation. The X-axis and Y-axis represent the concurrent flow-table consumption and PDF respectively, and a point (x, y) denotes that the y% switches are occupied by how much flow-entries. For example, in Fig. 7, there are about 90% switches taking less than 20 concurrent entries in peak in all network topologies case of PS. But for the ordinary operations, especially for the NSFNet scenario, the difference is more significant. There are more than 15% and 17% switches taking more than 80 and 90 concurrent entries in peak in the case of HT and IT, respectively. But, under the h = EF scenario, the advantage of the expedited invalid TCP flow eviction scheme is not very obvious (slight advantage). This is because the EF case lasts a too long time, hence the value of the timeout setting is much smaller than the duration of the EF, causing a communication duration almost the same as the value of timeout, thus resulting in such a situation.

Fig. 4(c) and 4(d) display similar results with Fig. 4(a) and 4(b), where the simulation results are about the expedited invalid SCTP flow eviction scheme and two ordinary flow expiry mechanisms. Unlike the simulation of flow-table consumptions at individual switches, in addition to the situation S2: 90%-10% PS curve, we also simulate that the situation S1: 80%-20% PS curve, where 80% of the SCTP communication has successfully established a connection, while the remaining 20% indicates that the SCTP-INIT message has been sent but failed to establish. As shown in Fig. 4(c) and 4(d), the performance of the expedited invalid SCTP flow eviction scheme is more excellent than the method based on TCP, especially under the h = EF scenario. This

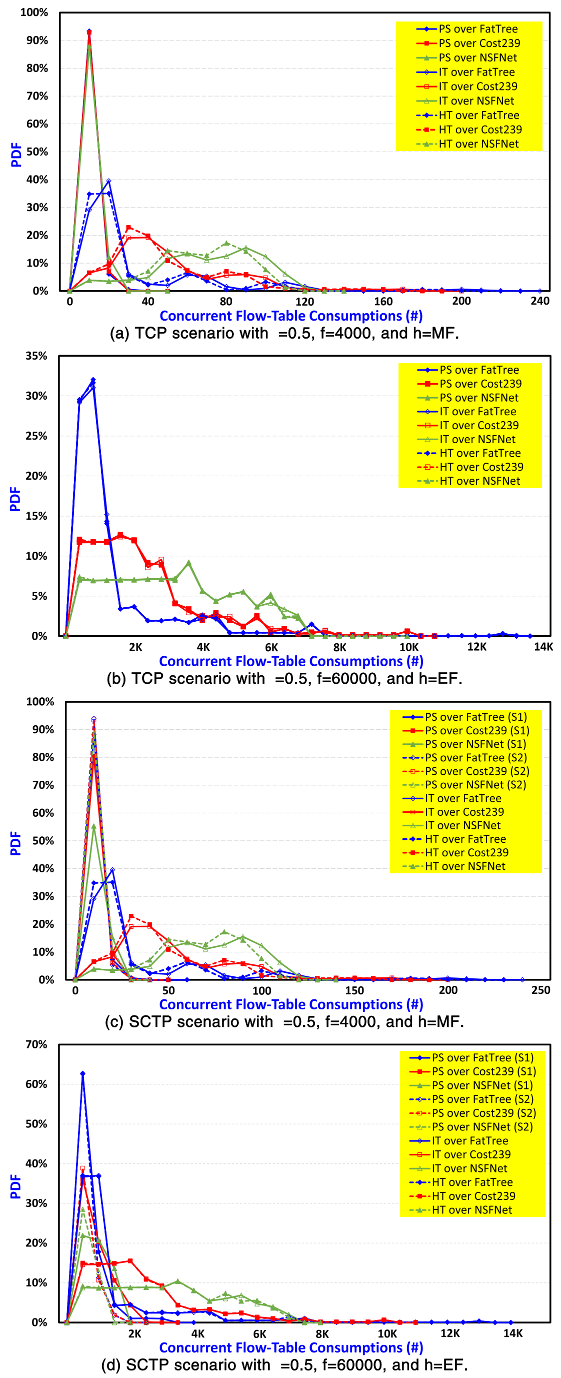


FIGURE 4. The PDF of concurrent flow-table consumptions of three competed schemes under three reference topologies in four scenarios.

is because the SCTP method adds the timeout mechanism and the heartbeat mechanism, which makes it more flexible to delete invalid flow-entries. As the Fig. 9 and 10 show, regardless of network topologies, it is obvious that all the curves of our scheme are mainly concentrated in the front part of the table and the value of the peak is quite high, which expresses a larger amount of switches occupied by a small number of flow-entries. But the results of the ordinary operations are not satisfying, which displays many switches are occupied by a large number of flow-entries. By means of comparing the flow-table consumption at individual switches, we can conclude that our scheme evicts the invalid flow-table more timely and quickly than the other ordinary operations.

D. FLOW-TABLE MISS RATES

As we know, the next incoming flow-entry cannot match the match fields in the switch in time due to the available entries of the flow-table is fully occupied, which will result in table-miss. Thus, evaluating the performance of the flow-table miss rate is very necessary for SDN.

Fig. 5(a) and 5(b) demonstrate the relationship between flow-table miss rate and different network traffic λ based on expedited invalid TCP flow eviction scheme and two ordinary flow expiry mechanisms in the scenarios of constant flow-table limitations Φ under $h = MF$ and $h = EF$ separately. As expected, the table-miss rate increases rapidly because a large amount of network traffic occupies the switch. But we can see that in the early stage of network traffic growth, our scheme can suppress the rapid growth of table-miss rate more than the other ordinary operations due to the mechanism which evicts the invalid TCP flow-entries quickly and can significantly slow the growth of table-miss rate under $h = MF$ scenario. It is worth noting that the effect of the EF simulation [Fig. 5(b)] is not obvious due to the long EF simulation time and the defects of the TCP mechanism, which we will further discuss in the next subsection.

Fig. 5(c) and (d) also depict the flow-table-miss rates in the given scenarios for different traffic intensities λ and $h = MF$ and EF respectively, based on expedited invalid SCTP flow eviction scheme and two ordinary flow expiry mechanisms in the scenarios of constant flow-table limitations Φ. From these two figures, it can be seen that no matter what percentages of SYN flooding attacks have, we can see that our method can effectively reduce the table-miss rate in the early stage and slow down the increase of the table-miss rate. In addition, the ordinary operations result in the rapidly exponential increase of table-miss rate in the early stages, which means that our method can evict invalid flow-tables faster, leaving more space for future flows matching.

Fig. 6 depict the flow-table-miss rates in the given scenarios for different flow-table limitations Φ and $h = MF$ and EF, and in the scenarios of constant network traffic, based on TCP and SCTP eviction schemes respectively. As shown in the figure, an increase in the number of flow-tables (flow-table limitation Φ) at switches will cause the table-miss rate to drop as expected. And it is obvious that our scheme can better achieve the goal of reducing the flow-table miss rate with given lower flow-table limitation (in addition to the expedited invalid TCP flow eviction scheme in the $h = EF$ scenario, the effect is not obvious because of the same reason). In contrast, the table-miss rates of ordinary operation decrease very slowly, and the table-miss rate only can achieve better results when the flow-table limitation is high enough. It is worth noting that some of the table missing rate curves in the simulation have obvious simulated inflection points. The reason is the

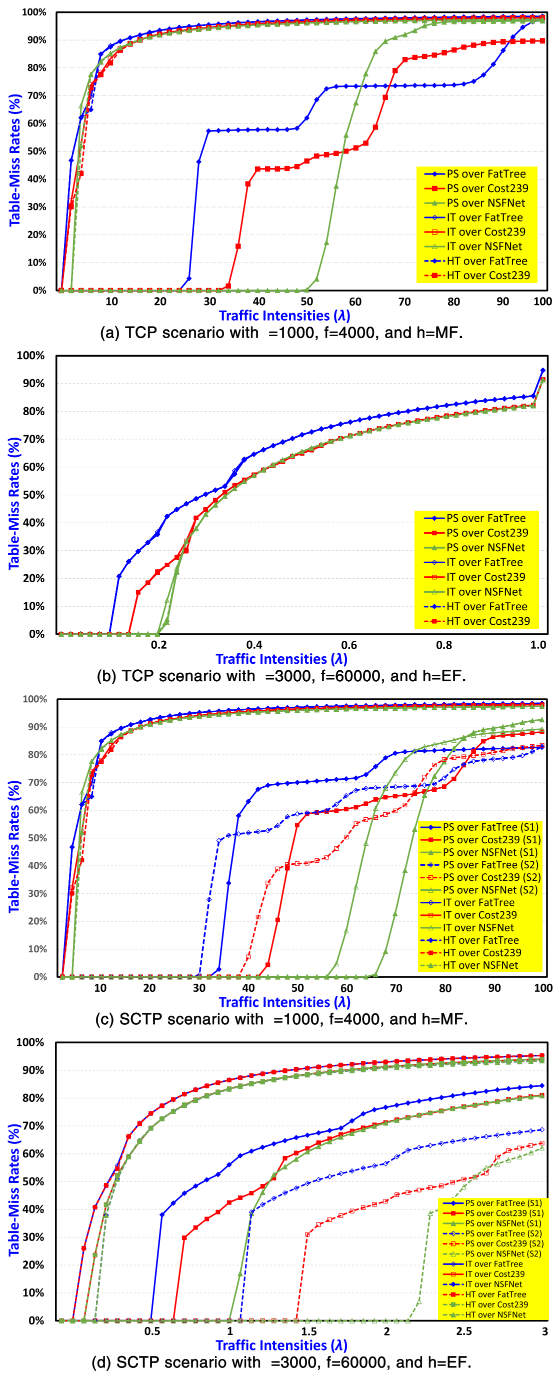
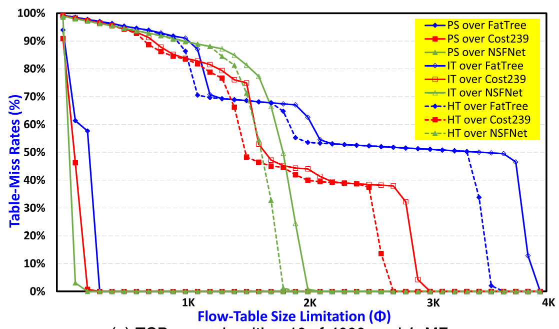


FIGURE 5. **Flow-table miss rate vs. traffic intensities of three competed schemes under three reference topologies in four scenarios.**

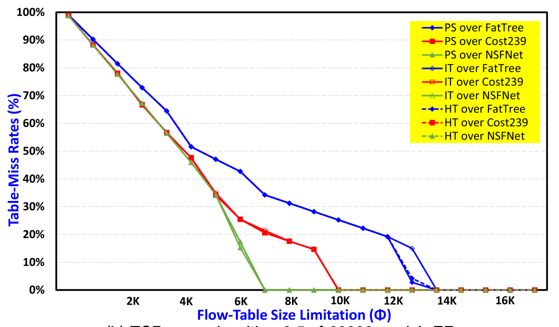
bottleneck effect—certain key nodes on the critical paths are accompanied by insufficient available space of flow-table that provides immediate performance degradations.

E. CONTROL OVERHEADS IN OPENFLOW-BASED SDNS

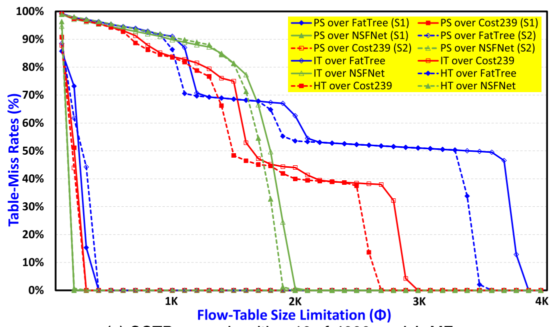
We also concerned the controller overhead costs between controller and switches to be inevitably increased by the proposed scheme, and performed a simulation experiment to evaluate the number of control messages and their bandwidth occupancies to be inevitably increased by the proposed



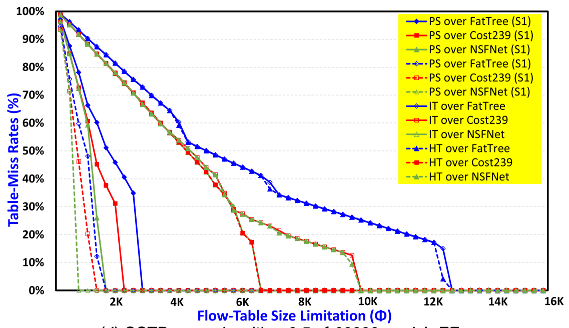
(a) TCP scenario with =10, *f*=4000, and *h*=MF.



(b) TCP scenario with =0.5, *f*=60000, and *h*=EF.

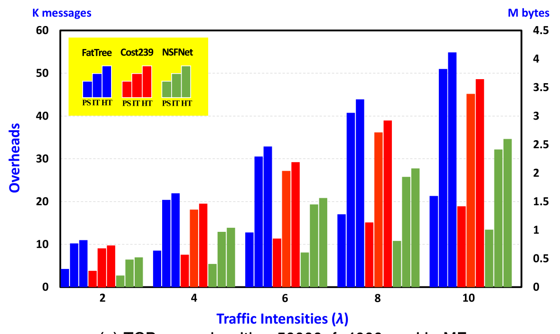


(c) SCTP scenario with =10, *f*=4000, and *h*=MF.

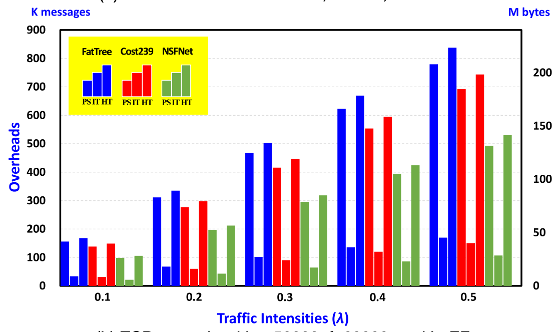


(d) SCTP scenario with =0.5, *f*=60000, and *h*=EF.

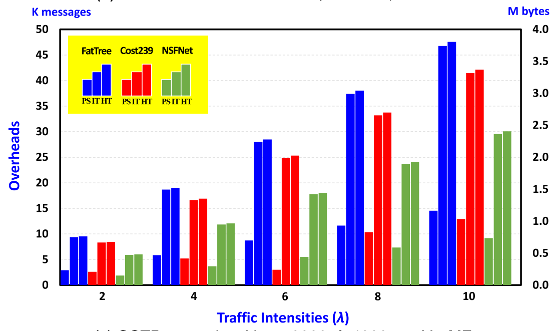
FIGURE 6. **Flow-table miss rate vs. flow-table size limitations of three competed schemes under three reference topologies in four scenarios.**



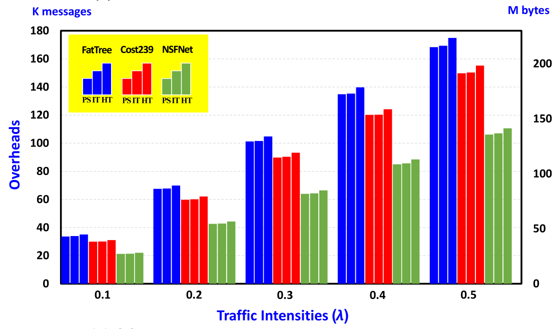
(a) TCP scenario with =50000, f=4000, and h=MF.



(b) TCP scenario with =50000, f=60000, and h=EF.



(c) SCTP scenario with =50000, f=4000, and h=MF.



(d) SCTP scenario with =50000, f=60000, and h=EF.

FIGURE 7. **Control overhead vs. traffic intensities of three competed schemes under three reference topologies in four scenarios.**

scheme. In the simulation of the traffic type h = MF and h = EF scenarios, a single flow with a sending rate $\lambda = 2, 4, \ldots, 10$ and $\lambda = 0.1, 0.2, \ldots, 0.5$ packets/second respectively were injected in the reference networks and for every f = 4000 and f = 60,000 packets respectively the number of control messages and their total datagram sizes were sampled. The measured control message includes `Packet-In`, `Packet-Out`, `Flow-Mod`, and `Flow-Removed` messages. All messages exclude the required control messages in OpenFlow (e.g., LLDP, ARP, ICMP, and other link-state advertisement packets) which appeared during the network uptime due to the topology discovery in the SDN controller. For standard OpenFlow protocol, the proposed scheme produces up to 15 and 11 control messages on each TCP and SCTP session respectively between controller and switch. For legacy flow expiry mechanisms, we suppose each initial connect-oriented connection produces at least 4 and up to innumerable control messages between controller and switch, and these are assumed to be the distinction as the average result of that on the idle_timeout and hard_timeout situations, when the session durations are long enough. Fig. 7 depicts a comparison of accumulated number of control messages and their bandwidth consumptions in bytes, which were measured in the aforementioned reference network topologies with similar conditions. The primary coordinate unit level is expressed in K messages, which is used to represent the accumulated number of control messages; the secondary coordinate unit level is expressed in M bytes, which is used to display the control bandwidth consumptions.

As seen in Fig. 7(b), in the TCP with the MF scenario, the control overhead of the proposed scheme is significantly less than that of the other two existing flow expiry mechanisms, especially better than the inflexible HT method. In the TCP with the EF scenario, the signaling overhead of the method in this article is more than that of the IT method, and is significantly less than that of the HT method. This is because of the long duration connections; the half-closed state of the fast invalid TCP flow eviction scheme causes massive overheads. As a result, it is not as good as the IT method. This situation will be resolved by using the P4 switch technology, but overall, the proposed scheme is acceptable in terms of control overhead.

As shown in Fig. 7(c) and 7(d), in the two SCTP scenarios, the control overhead of the proposed scheme is not high. The IT method also does not cause too much signaling control due to its more flexible characteristics. The feature of the HT method causes very high signaling overhead, especially in EF scenarios. That is because of the HT method deleting flow entries at an always imperfect constant expiry time interval. Thus the fast invalid SCTP eviction scheme is superior to the legacy flow expiry mechanisms in terms of signaling overheads, and does not cause excessive control overheads.

From these results, it is mainly noted that once the proposed approach has implemented by P4 technology in switch-side, by decentralizing the network's intelligence toward the edge in SDNs, it will further succeed in reducing the number of control messages such as `Packet-In`, `Packet-Out`, and `Flow-Removed`, etc.

V. CONCLUSION

In this article, we revisit the reasons for causing the scalability issues of SDN, since the size of an SDN switch's TCAM is finite and the ordinary operations are not timely enough to evict the invalid flow-entries. To improve the scalability, we propose a scheme to achieve early detecting and evicting the invalid flow-entries of connection-oriented protocols immediately and minimize the concurrent flow-table occupancies in SDN switches. The scheme can be divided into two parts: the countermeasures for the TCP and SCTP, called the expedited invalid TCP flow eviction scheme and the expedited invalid SCTP flow eviction scheme, respectively.

A series of simulations based on practical SDN parameters are used to evaluate the performance of the proposed scheme and the results prove that our scheme can achieve the purpose of quickly evicting invalid flow-entries and significantly improve the scalability of SDN, that corresponds to intuition. However, the expedited invalid TCP flow eviction scheme in the elephant traffic scenario does not achieve very good results, which needs to be improved in our future work. In addition, the lifetime of different port numbers under normal conditions will also depend on the traffic characteristics of that particular application. In the follow-up study, we can consider checking the port number with SDN rulesets, and assigning different initial idle timeout values to different port numbers or generating dynamic adaptive and dynamically adjustable idle timeout initial values via machine learning. It is expected that this method will be further optimized and this part is for further study.

Undeniably, a certain of SDN control messages are generated each time a TCP/SCTP control message appears which is very common in network traffics increasing the control overhead and decreasing the feasibility of the proposed scheme in the ordinary Openflow-based SDNs. Positively, the proposed scheme in some cases, which may be embedded into the SDN switches, takes over the responsibilities of the TCP/SCTP control signaling detector and consolidate proposed functions into a new feature set to be implemented on the newer P4 switches. The improved P4 switch checks the concurrent number of connections for both the transport-layer protocols and evicts the invalid flow-entry right after the last data flow is away by detecting the control messages of the famous connection-oriented protocols. The above-mentioned action to be accomplished within zero waiting time. Thus, scalability in terms of the available flow-table of the SDNs increases with quite little additional costs.

REFERENCES

[1] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, and J. C. Zhang, "What will 5G be?" *IEEE J. Sel. Areas Commun.*, vol. 32, no. 6, pp. 1065–1082, Jun. 2014.

[2] C. Elliott, "GENI: Opening up new classes of experiments in global networking: Internet predictions," *IEEE Internet Comput.*, vol. 14, no. 1, pp. 39–42, Jan. 2010.

[3] A. Gavras, A. Karila, S. Fdida, M. May, and M. Potts, "Future Internet research and experimentation: The FIRE initiative," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 3, pp. 89–92, Jul. 2010.

[4] L. Yang, R. Dantu, T. Anderson, and R. Gopal, *Forwarding and Control Element Separation (ForCES) Framework*, document IETF RFC-3746, Apr. 2004.

[5] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, pp. 41–54, Oct. 2005.

[6] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in *Proc. ACM NSDI*, Berkeley, CA, USA, vol. 2, May 2005, pp. 15–28.

[7] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "SANE: A protection architecture for enterprise networks," in *Proc. ACM USENIX-SS*, Vancouver, BC, Canada, Aug. 2006, vol. 15, no. 10, p. 50.

[8] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *Proc. ACM SIGCOMM*, Aug. 2007, pp. 1–12.

[9] (Mar. 2015). *OpenFlow Switch Specification, Version 1.5.1*. [Online]. Available: https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/OpenFlow-switch-v1.5.1.pdf

[10] A. Yassine, H. Rahimi, and S. Shirmohammadi, "Software defined network traffic measurement: Current trends and challenges," *IEEE Instrum. Meas. Mag.*, vol. 18, no. 2, pp. 42–50, Apr. 2015.
[11] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 136–141, Feb. 2013.
[12] S. Luo, H. Yu, and L. Li, "Practical flow table aggregation in SDN," *Comput. Netw.*, vol. 92, pp. 72–88, Dec. 2015.
[13] S. Veeramani, M. Kumar, and S. N. Mahammad, "Minimization of flow table for TCAM based openflow switches by virtual compression approach," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst. (ANTS)*, Kattankulathur, India Dec. 2013, pp. 1–4.
[14] S. Saraswat, V. Agarwal, H. P. Gupta, R. Mishra, A. Gupta, and T. Dutta, "Challenges and solutions in software defined networking: A survey," *J. Netw. Comput. Appl.*, vol. 141, pp. 23–58, Sep. 2019.
[15] R. Challa, Y. Lee, and H. Choo, "Intelligent eviction strategy for efficient flow table management in OpenFlow switches," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Seoul, South Korea, Jun. 2016, pp. 312–318.
[16] M. Rifai, D. Lopez-Pacheco, and G. Urvoy-Keller, "Coarse-grained scheduling with software-defined networking switches," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 95–96, Aug. 2015.
[17] H. Zhu, H. Fan, X. Luo and Y. Jin, "Intelligent timeout master: Dynamic timeout for SDN-based data centers," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM*, Ottawa, ON, Canada, May 2015, pp. 734–737.
[18] *OpenFlow Timeouts*. Accessed: Oct. 2018. [Online]. Available: https://www.sdnlab.com/22563.html
[19] N. Gude, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.
[20] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," *Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.
[21] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 351–362, Aug. 2010.
[22] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite CacheFlow in software-defined networks," in *Proc. ACM HotSDN*, Chicago, IL, USA, Aug. 2014, pp. 175–180.
[23] T. Kim, K. Lee, J. Lee, S. Park, Y. H. Kim, and B. Lee, "A dynamic timeout control algorithm in software defined networks," *Int. J. Future Comput. Commun.*, vol. 3, no. 5, pp. 331–336, Oct. 2014.
[24] K. Kannan and S. Banerjee, "FlowMaster: Early eviction of dead flow on SDN switches," in *Proc. Int. Conf. Distrib. Comput. Netw. (ICDCN)*, Berlin, Germany, Jan. 2014, pp. 484–498.
[25] A. Vishnoi, R. Poddar, V. Mann, and S. Bhattacharya, "Effective switch memory management in OpenFlow networks," in *Proc. 8th ACM Int. Conf. Distrib. Event-Based Syst. (DEBS)*, Mumbai, India, May 2014, pp. 177–188.
[26] E.-D. Kim, Y. Choi, S.-I. Lee, and H. J. Kim, "Enhanced flow table management scheme with an LRU-based caching algorithm for SDN," *IEEE Access*, vol. 5, pp. 25555–25564, 2017.
[27] Z. Guo, R. Liu, Y. Xu, A. Gushchin, A. Walid, and H. J. Chao, "STAR: Preventing flow-table overflow in software-defined networks," *Comput. Netw.*, vol. 125, pp. 15–25, Oct. 2017.
[28] Z. Guo, Y. Xu, R. Liu, A. Gushchin, K.-Y. Chen, A. Walid, and H. J. Chao, "Balancing flow table occupancy and link utilization in software-defined networks," *Future Gener. Comput. Syst.*, vol. 89, pp. 213–223, Dec. 2018.
[29] A. Panda, S. S. Samal, A. K. Turuk, A. Panda, and V. C. Venkatesh, "Dynamic hard timeout based flow table management in openflow enabled SDN," in *Proc. Int. Conf. Vis. Towards Emerg. Trends Commun. Netw. (ViTECoN)*, Vellore, India, Mar. 2019, pp. 1–6.
[30] S. Banerjee and K. Kannan, "Tag-in-tag: Efficient flow-table management in SDN switches," in *Proc. 10th Int. Conf. Netw. Service Manage. (CNSM)*, Rio de Janeiro, Brazil, Nov. 2014, pp. 109–117.
[31] Y. Lu, Z. Ling, S. Zhu, and L. Tang, "SDTCP: Towards datacenter TCP congestion control with SDN for IoT applications," *Sensors*, vol. 17, no. 12, p. 109, Jan. 2017.
[32] S. Shirali-Shahreza and Y. Ganjali, "Delayed installation and expedited eviction: An alternative approach to reduce flow table occupancy in SDN switches," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1547–1561, Aug. 2018.
[33] C.-H. He, B. Y. Chang, S. Chakraborty, C. Chen, and L. C. Wang, "A zero flow entry expiration timeout p4 switch," in *Proc. Symp. SDN Res.*, New York, NY, USA Mar. 2018, pp. 1–2.
[34] R. Ying, W.-K. Jia, Y. Zheng, and Y. Wu, "Fast invalid TCP flow removal scheme for improving SDN scalability," in *Proc. 16th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, Las Vegas, NV, USA, Jan. 2019, pp. 1–5.
[35] R. Ying, W.-K. Jia, C. Luo, and Y. Wu, "Expedited eviction of invalid flow entries for SDN-based EPC networks," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, Changchun, China, Aug. 2019, pp. 298–303.
[36] R. Stewart, *Stream Control Stream Transmission Protocol*, document IETF RFC-2960, Sep. 2007.
[37] A. L. Caro, J. R. Iyengar, P. D. Amer, S. Ladha, G. J. Heinz, and K. C. Shah, "SCTP: A proposed standard for robust Internet data transport," *Computer*, vol. 36, no. 11, pp. 56–63, Nov. 2003.
[38] (Apr. 2013). *OpenFlow Switch Specifications, Version 1.3.2*. [Online]. Available: https://www.cs.princeton.edu/courses/archive/fall13/cos597E/papers/OpenFlow-spec-v1.3.2.pdf

WEN-KANG JIA (Senior Member, IEEE) received the Ph.D. degree from the Department of Computer Science, National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 2011. Before returned to school, he has been a Senior Engineer and a Manager since 1991 in various networking areas including ICT Manufacturer, Network Integrator, and Telecomm Service Provider. Since January 2018, he has been a Full Professor of the College of Photonic and Electronic Engineering (P&EE), Fujian Normal University (FJNU), Fuzhou, China. He has published more than 100 research articles, which has been cited more than 500 times. His current research interests include OSI layer-2/3/4 such as TCP/IP protocol design, high-performance switching and routing, multicasting and broadcasting, mobile management, error resilience coding, multimedia communications, QoS and teletraffic engineering, IP-optical convergence networks, P2P overlay networks, cloud computing, and 4G/5G mobile networks. He was awarded the 2nd Fujian Province Hundred Talent Plan, in 2019.

RUOLAN YING received the M.S. degree from Fujian Normal University (FJNU), Fuzhou, Fujian, China, in 2020, where she is currently pursuing the master's degree with the College of Photonic and Electronic Engineering (P&EE). She has been a Patent Examiner of Communication Engineering Room, Patent Examination Cooperation, Beijing, Fujian, Center of the Office, which is under the jurisdiction of China National Intellectual Property Administration (CNIPA), Fujian, China, since 2020. Her current research interests include network protocol designs, network routing and forwarding, and emerging network architectures.

XIAONING SHI received the B.S. degree from the School of Business Administration, Jimei University, Xiamen, Fujian, China, in 2002. Since 2020, she has been a Research Assistant with the College of Photonic and Electronic Engineering (P&EE), Fujian Normal University (FJNU), Fuzhou, China. Her current research interests include social networks, P2P networks, multimedia networks, cloud and edge computing, and electronic commerce.

• • •