*Review*

# Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey

**Babangida Isyaku** [1,2,*] **, Mohd Soperi Mohd Zahid** [3] **, Maznah Bte Kamat** [1] **, Kamalrulnizam Abu Bakar** [1] **and Fuad A. Ghaleb** [1]

[1]  Department of Computer Science, Faculty of Engineering, Universiti Teknologi Malaysia, Johor Bahru 81310, Malaysia; kmaznah@utm.my (M.B.K.); knizam@utm.my (K.A.B.); aafuad@utm.my (F.A.G.)

[2]  Department of Mathematics and Computer Science, Sule Lamido University, Kafin Hausa P.M.B 048, Jigawa State, Nigeria

[3]  Department of Computer and Information Science, Universiti Teknologi PETRONAS, Seri, Iskandar Perak 32610, Malaysia; msoperi.mzahid@utp.edu.my

[*]  Correspondence: isyaku@graduate.utm.my; Tel.: +60-1161-110-962

check for updates

**Abstract:** Software defined networking (SDN) is an emerging network paradigm that decouples the control plane from the data plane. The data plane is composed of forwarding elements called switches and the control plane is composed of controllers. SDN is gaining popularity from industry and academics due to its advantages such as centralized, flexible, and programmable network management. The increasing number of traffics due to the proliferation of the Internet of Thing (IoT) devices may result in two problems: (1) increased processing load of the controller, and (2) insufficient space in the switches' flow table to accommodate the flow entries. These problems may cause undesired network behavior and unstable network performance, especially in large-scale networks. Many solutions have been proposed to improve the management of the flow table, reducing controller processing load, and mitigating security threats and vulnerabilities on the controllers and switches. This paper provides comprehensive surveys of existing schemes to ensure SDN meets the quality of service (QoS) demands of various applications and cloud services. Finally, potential future research directions are identified and discussed such as management of flow table using machine learning.

**Keywords:** SDN; OpenFlow; rule management; update operation; security threat

## 1. Introduction

The number of internet-connected or Internet of Things (IoT) devices, and traffic flow volume in the internet has significantly increased and continues to grow. According to Pierre et al. [1], the number of active internet-connected devices was 26.66 billion in 2019. The annual global network traffic flow as of 2018 amounted to 19.01 exabytes per month and is expected to reach 77.5 exabytes per month by 2022 worldwide [1]. The data flows are coming mostly from IoT devices and stored in data centers, which provide various and increasing kinds of cloud services to users. Managing the internet and data centers to meet the expanding demands of emerging applications such as real-time has become a big challenge in traditional networking. The integration of control and forwarding logic in routers and switches in traditional networking introduces some limitations and inflexibility in managing and monitoring the networks to configure the optimum quality of service (QoS) provisioning. The configuration of policies (e.g., firewall and routing policies) required network operators to adhere to the vendor-specific interfaces of routers and switches to configure the device [1]. Adjustments to the network have to be made by the network administrator more conveniently and promptly to cope

with dynamic network behaviors such as fault management and load changes [2]. Despite the wide adoption of internet protocol (IP) based networks, automatic reconfiguration and response to network changes are difficult in the current conventional networks, according to Kreutz et al. [2].

SDN decouples the control plane (networking logic) from the data plane (forwarding logic) [3]. The control plane consists of controllers acting as a network operating system and the data plane consists of switches, which mainly forward packets to the next hop. The controller is responsible for maintaining the global network state in real-time. The separation between control and data planes is achieved using a standard communication interface such as OpenFlow [4]. OpenFlow switches have Flow Table, which is commonly implemented with Ternary Content Addressable Memory (TCAM) technology. Typically, a flowtable is populated with rules or policies such as quality of service (QoS), access control lists (ACLs), and IP route tables for fast-forwarding. However, TCAM is power-hungry, expensive, and available in limited space or capacity that can only accommodate from 750 to 20,000 flow entries [5,6]. Thus, the flow table is relatively small compared to the number of required rules. The centralized architecture of SDN and space limitations of the flow table introduce some performance and security issues. Attackers may overwhelm the flow table with multiple denial of service (DoS) attacks and manipulate the controller to refuse writing legitimate flow entries [7]. Mitigating the DoS attacks requires the installation of a large number of access control list (ACL) distributed rules while TCAM is available in only limited capacity or space.

SDN controller populates or updates the flow table reactively or proactively upon occurrence of some events. In a reactive approach, the controller does not populate the flow table with any rules when network operation begins. Whenever packets arrive at switches during network operation, rules will be installed into the flow table by the controller. For the proactive approach, the controller will install flow entries in the flow table in advance, when network operation begins. The selection of rules is imperative in maximizing network performance especially in large-scale networks such as data centers. When packets arrive at a switch during network operation, the flow of arriving packet is matched against flow entries in the flow table. If matching is not found, the switch will contact the controller to update the flow table with entries that allow the packet to reach its destination. This involves communication overhead between controller and switch as well as delay until the packet can be forwarded to the next hop. The proactive approach was introduced to reduce the communication overhead involved between switches and controllers.

Another SDN performance issue is regarding the flow table update operation made upon the occurrence of events such as topology changes [5], network reconfiguration [6], and the creation of re-routing rules [7]. The routing update operation must be completed within 25 milliseconds (ms) to meet stringent QoS requirements of real-time applications [8]. Similarly, failure recovery has a strict recovery delay requirement to adhere to the carrier-grade quality [8]. Fast rules rerouting depends on switch rule updating time and controller response time to generate rules. A common challenge experienced by the SDN controller is to update the switches consistently and promptly. The longer time the update operation takes, the more probability for the network to have unstable behaviors such as extra packet processing delay, forwarding loop, and routing errors [7]. Thus, the centralized controller, flow table limitation of commercial switches, and flow table update operation should be dealt with efficiently to avoid critical performance bottleneck in the deployed production network environment of OpenFlow-SDN [5]. These three issues of SDN have gained great attention from researchers in recent years [7–11]. DevoFlow [9], DIFANE [12], and Kotani [13] proposed some schemes to reduce the processing load of the controller. SDN-Guard [14–16] have proposed some methods to mitigate security attacks arisen due to the centralized nature of the controller in SDN. Some solutions were proposed to reduce higher flowtable update operations [7,17,18]. Other solutions focus on improving the efficiency of a limited flowtable [19–21].

This paper provides a comprehensive survey on the efforts that have been done on optimizing the processing load of the controller, mitigating malicious attacks, flow table update operation, and improving the efficiency of flow table management to ensure stable performance of SDN.

Potential future research directions are also identified and discussed, such as the use of machine learning and other artificial intelligence techniques. The paper road map is organized as follows: Section 2 presents an overview of OpenFlow as a promising standard to achieve the benefit of SDN. Section 3 explains SDN performance challenges. Section 4 presents and details the existing state-of-the-art proposed solutions to manage flow table memory. Section 5 discusses and highlights the research challenges and suggests future research direction. Section 6 rounds up the paper and presents concluding remarks.

## 2. Overview of SDN Architecture

SDN consists of three planes: application plane (AP), control plane (CP), and data plane (DP), as depicted in Figure 1. Application planes consist of network applications such as network virtualization, firewalls, intrusion detection system (IDS), and mobility management that leverage the exposed northbound (NB) application interface to interact with the control plane. The NB interface provides a stable, consistent way for network administrators and application developers to efficiently use SDN services to implement important network management at the control plane. The CP is the most essential part of the SDN structure, it provides fine-grained control over the networking element at the DP and offers many network services, which include routing computation, monitoring, load balancing. The CP translates these services from application-level into a clear set of instructions and requests in form of flow entries and installs them in the corresponding switch devices. Single CP can be configured for the entire network, but for scalability reasons, it can be extended to distributed or multiple controllers. As for the DP, they are used as a simpler forwarding element with no software capable of making an instant decision. Therefore, network intelligence is withdrawn and shifted to a centrally logical controller. For any control decision, the switch must consult the controller for further action. This paper focus on the control plane and data plane.
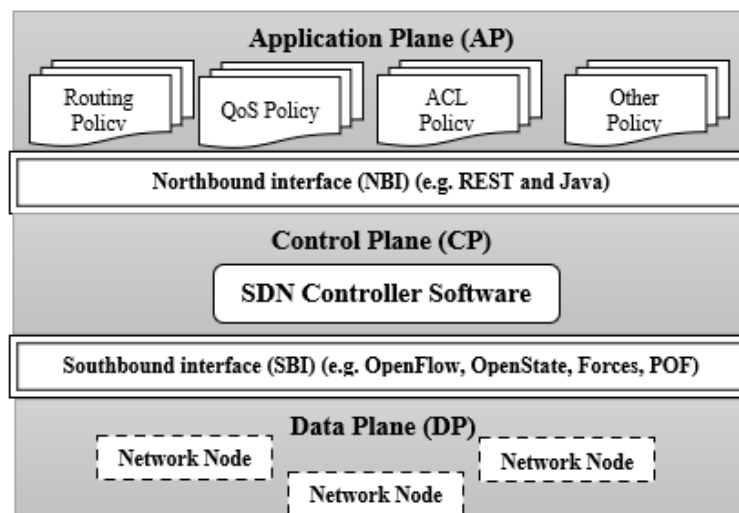


**Figure 1.** Software defined networking architecture.

### 2.1. SDN Southbound Interface

OpenFlow is so far the most popular standard for SDN southbound (SB) interface. Earlier implementations of SB interface were forwarding and control elements (ForCES) and protocol-oblivious forwarding (POF) [2]. ForCES uses logical function blocks (LFB) in the data forwarding elements to provide networking functionalities such as IP routing [3]. However, these standards rely essentially on modifying forwarding devices to support flow tables [2], in a way that can be dynamically configured by remote entities through operations such as, adding, removing, or updating flow rules in the flow table. Therefore, a recent initiative such as OpenFlow emerges,

which does not require modifications of the switches to support flowtable. This attracts not only the research community but also the networking industry [2]. OpenFlow became the most popular and powerful implementation of the SB interface [22], standardized by Open Network Foundation [23] to control the behavior of the SDN devices. [24] developed OpenState as a superset extension of Openflow aimed at offloading some of the control logic to switches thereby shifting the pragmatic approach from stateless to stateful operation. This means the OpenFlow switches can also be directly programmable, thus they can handle forwarding rules locally without the need to completely depend on the remote controller. OpenState is yet to be implemented as the future extended version of OpenFlow [25]. In OpenState, the legacy flow table in OpenFlow is preceded by a state table to processed flow states, by using flow-states and global states. A programmer can define flow entries that apply to different scenarios, using state transition, a programmer can control the evolvement of different scenarios. This survey focuses on OpenFlow as the standard southbound interface between switch to SDN Controller.

## 2.2. Flow Table of SDN Switches

In SDN, forwarding entities are referred to as OpenFlow switches and all forwarding decisions are flow-based instead of destination-based as used in the legacy traditional network. An OpenFlow switch contains a flow table with a logical data structure where packets are processed based on a list of prioritizing entries. A flow table can store a set of flow entries made of 15 field tuples in OpenFlow 1.10: some of the fields are optional but matching fields, action, statistical counter, priority, and timeout mechanism are commonly used. Matching fields are used to match packet meta information such as (medium access control (MAC) source and destination address, ethernet type, internet protocol (IP) source and destination address). These fields can be specified as either exact-match or wildcard-match entries. The former represents an individual flow, while the latter represent multiples flows using any value inform of an asterisk (*). Incoming packets will be looked up in the flowtable, if the packet matches either exact-match or wildcard flow, the corresponding action will be taken. The statistical count will be incremented for all successful matches of flow entry with the arriving packet. Wildcard matching entries are assigned with priorities, if multiple packets match multiple wildcards flow the higher priority is the final match. Exact match is usually assigned a higher priority than wildcard flows. If the packet could not match any flow, it will be forwarded to the controller or drop the packet. Other actions may be forward to specific switch port number. The flow lookup is compulsory for every incoming packet. This could be done using a single flowtable or multiple flowtable pipelining up to 255 tables [26] as illustrated in Figure 2:

For efficiency and flexibility reasons, OpenFlow 1.3 [26,27] supports multiple flow table pipelining processing where a packet may be processed by more than one flow table. Both the single and pipelining tables have the four components (matching field, action, statistic, priority, and timeout). In contrast to the single table, the matching process starts at the first flow table in the pipelining tables and may continue to the next table until the match flow entry is found. The process of checking entry is performed sequentially, and action is normally executed at the end of the last table in the pipelining.

If there is no match, the switch may either drop the packet or trigger a *packet-in* message to the controller for flow setup request depending on the configuration of the table-miss entry. In the event of a packet forwarded to the controller for flow setup request, the centralized controller computes new flow entry and sends a packet-out event to instruct the switch to install the corresponding entry in its flowtable. Frequent flow setup requests will be sent to the controller on a large scale with the presence of massive flow arrivals. According to [27], there can be up to 200,000 flows arrival/sec for a data center with a 4 k server. Another previous study report on average flow size shown to have around 20 packets per flow with flow inter-arrival time of less than 30 ms [28]. These demands are very high; however, the memory capacity to store forwarding entries is small. Typical OpenFlow switch flow tables stored rules in a special type of high-speed memory called ternary content addressable memory (TCAM), which provides constant flow entry lookup within a single clock cycle O (1). Despite the

high-speed lookup, TCAM's memory is constrained with a few thousand entries [26]. Increasing the TCAM size introduces another concern such as cost and will require high-power consumption.
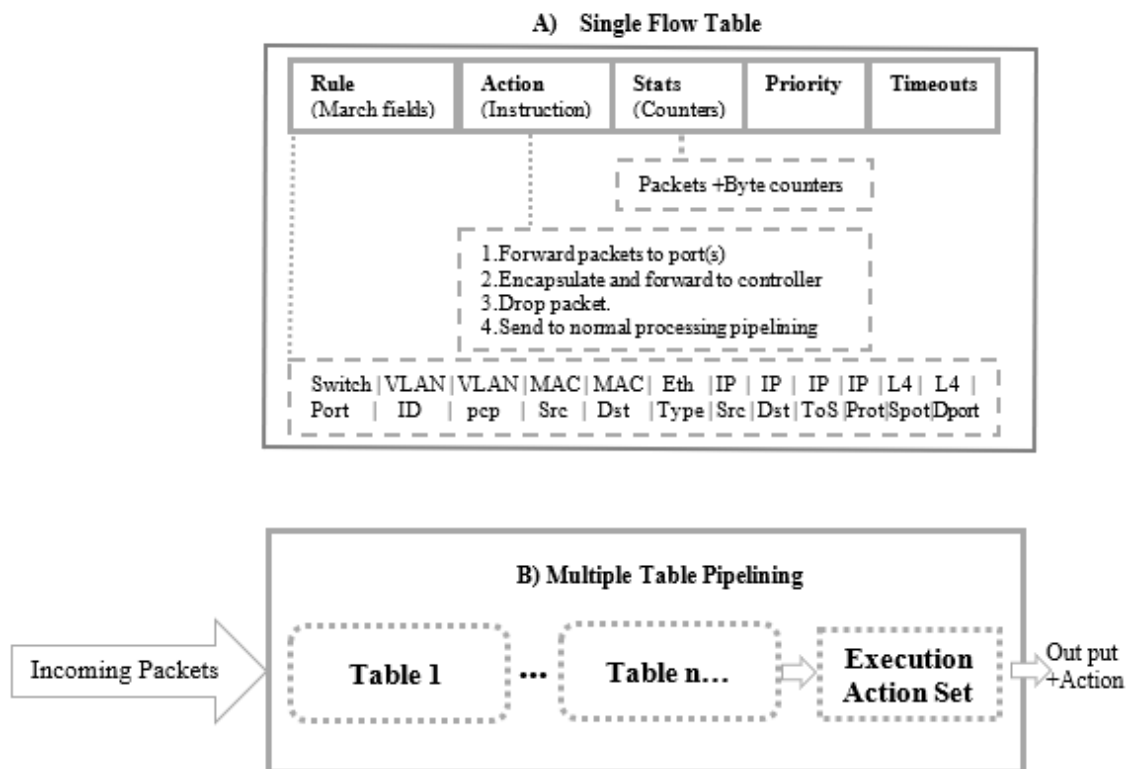


**Figure 2.** Single vs. multiple flow table. (**A**) Single Flow Table and (**B**) Multiple Flow Table respectively.

Lately, software switches built on commodity servers, which are part of the same switch hardware, are becoming popular [29]. This type of switch can offer a large flow table capacity with a high packet processing rate of 40 Gbps on a quad-core machine [29]. However, such switches are constrained in the lookup rate compared to commodity switches [30]. The software switch is designed based on the general-purpose central processing unit (CPU). In contrast, commodity switch is based on application-specific integrated circuit (ASIC), which is purposely designed for high-speed lookup [31]. In addition, software switches stored forwarding rules in the conventional random-access memory (RAM), which has a relatively lower cost with high storage capacity. OpenFlow switches store forwarding rules in TCAM, which is designed for matching flexibility and high lookup performance. Moreover, as explained by [32], installing forwarding rules in software switch instead of ASIC reduces the OpenFlow switch performance from 940 to 14 Mpbs. To further speedup switching operation in software switches, it is possible to store a flow table in CPU caches. However, these caches inherit similar shortcomings of storage limitation, which was introduced in ASIC [32]. Consequently, the shortage introduces a lot of problems, which include but is not limited to extra packet processing delay, update operation, and signaling overhead. Therefore, the switch flowtable limitation and frequent flow setup request to controller present great concern, which further needs to be investigated, especially in a large scales network environment such as data center and IoT. Section 4 extensively surveys the existing solutions proposed in the literature to cope with the limitations of the switch flow table memory.

### 2.3. Installation of Flow Table Entries

Normally, flow entries are installed in the switch flow table in two modes: reactive and proactive approaches. Similarly, occurrences of some events such as link failure or change in ACL rules may cause the flow table to be updated. Such a situation may require two operations, the old flow entries

need to be deleted and the new entry for the alternative flow or ACL rules respectively need to be installed. In this situation, flow table update operation is necessary. Section 3.2 presents flow table update operation, while the following sections detail the two approaches and performance issues associated with each approach were also discussed.

2.3.1. Reactive Flow Table Allocation

Initially, during the network booting time, switch flowtable is empty. Features negotiation between switch and controller will immediately commence for the controller to discover all the switches connected. There are two situations where the flowtable will be populated. The first situation is when the first packet of flows arrives in the switch flowtable. This will cause table-miss because of the unavailability of the entry. To handle the table-miss, switch performs reactive process through a packet-in event and sends it to the controller for further decision. The controller computes the correspondent entry of the flow and updates the switch flowtable of the affected switch. The second situation is re-installing of flow entries due to flow expiration or flow modification event. Generally, setting new flow requires 2 or n messages for many flows between the switch to the controller, where $n$ represents the number of data flows. Packets transmission commence after the rules have been installed. A subsequent packet of the same flow will be processed without consulting the controller. Figure 3 illustrates reactive packet processing. Assume 3 different packets belonging to different flows arrived at switch $S_1$ at different time intervals. Packet $p_1$ arrives at time $t_1$, packet $p_2$ arrives at time $t_2$ slightly after $t_1$, and packet $p_3$ arrives at time $t_3$ slightly after $t_3$. The switch performs 3 lookups in its flowtable after the arrival of packet $p_3$. Switch $S_1$ buffers the 3 packets and generates 3 number of a packet-in event to request the corresponding flows from the controller as indicated by arrow 1. Upon receiving each packet-in, the controller computes the flow entries and send a packet-out message back to switch $S_1$ instructing the switch to install corresponding entries as shown by arrow 2. Finally, packets are enqueued and transmitted at full line rate as shown by message 3.
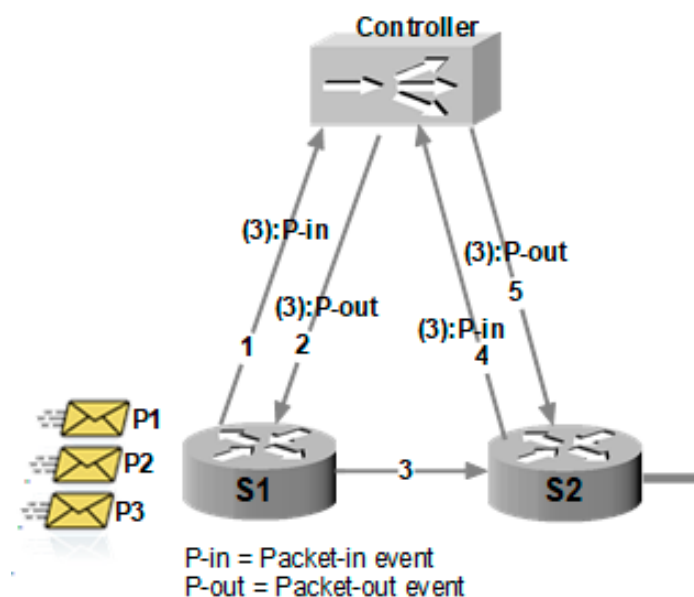


**Figure 3.** Example of reactive entry installation.

Intuitively, to adjust with the current network state, $n$ number of OpenFlow messages are exchanged between the two switches to the controller to get flow installed. This can easily decline the performance of the overall network and degrade the quality of service. Prior study in Liu et al. [28] reveal that initiating flow setup request for every new incoming flow together with the TCAM update operation increases the packet processing delay. For instance, upon occurrences of the link failure event, the controller is expected to reroute all affected flows within 25 milliseconds. The strict delay

requirement is another bottleneck for real-time application in SDN. Packets must be delivered to the desired destination with less delay guaranteed to avert declining QoS. This is necessary in order to adhere to the fine-grained traffic engineering delay requirement by CGN. A long delay will result in traffic congestion and increases packet losses, which can affect the performance of the network. In a study by Sharma et al. [29], the experiment results carried on the small-scale network with (6 and 14 switches) showed the possibility to meet the demand of carrier grade network CGN delay requirement. The same experiment reported that the delay also depends on the number of flow rules needed to modify or install in the affected switch. In the worst case, time can reach up to 300 milliseconds in 14 switches topology setting. Hence, it is challenging to meet the demand of the (CGN) delay budget in large scale networks. Additionally, the presence of a large number of flows introduced an extra processing load on both the controller and switch buffer. Therefore, using a reactive approach for all flows setting may not give the desire performance in large scale networks.

### 2.3.2. Proactive Flow Table Allocation

In contrast to reactive, proactive handles the incoming packet faster because the controller can install forwarding rules in advance before the flow arrival. Incoming flow whose packet headers information matches an existing entry is processed according to the instruction defined in the flow table, hence, the switch does not need to consult the controller to process the packet. For example, in Figure 4 the controller installed $n$ number of flow entries in advance before the arrival of packets as shown in arrow 1 and 2. In contrast, to the $n$ number of OpenFlow messages exchanged between the switch to the controller in reactive, the proactive approach reduced the messages. To some extent, the approach had improved the user QoS with less processing load on the controller. A prior study by Fernandez et al. [30] verified the performance of proactive rule installation, the experimental results outperform reactive approach in terms of throughout (flows per second) with various scenarios. The performance gain was attributed to zero additional flow setup time. Importantly, traffic flow will not be disrupted even when switch loses connection with the controller. Therefore, a proactive method can be preferable in achieving low packet processing delay with an acceptable performance where the flow table storage space is enough. However, it may not give better performance in SDN because of the switch memory constraint. Assuming 200 packets for different flows arrived at the switch flow table at the same time whose flowtable size is less than 200, in such a case, the chances of flowtable overflow are unavoidable. Consequently, it may lead to the removal of flow that belongs to active flow. Future packets meant for active flow, forces switch to trigger more packet-in event to the controller to repeatedly update the flowtable for the same flow entry.
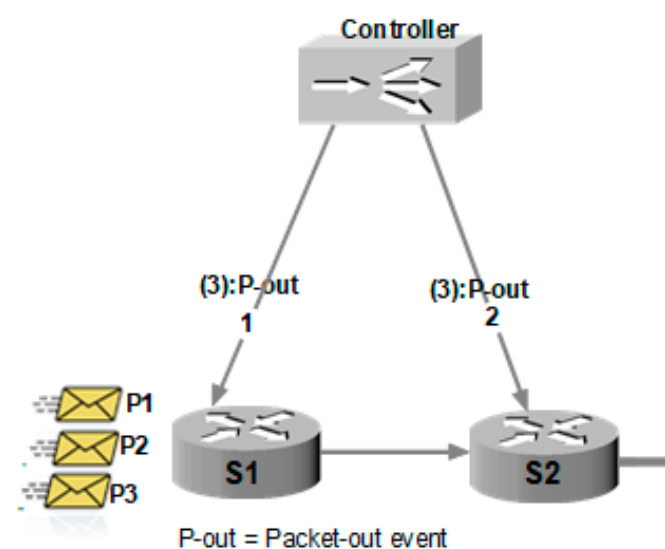


**Figure 4.** Example of proactive entry installation.

The significant traffic increase and small TCAM space gap have remained one of the challenging issues in a proactive approach. This challenge is more critical for large scale networks, for example in the event of failure, a huge number of backup flow entries required besides primary path flow entries to guarantee the survivability of the network state. Pushing extra flow entries can easily overwhelm the TCAM and chances of overflow are unavoidable. Additionally, flow entry update operation is hard with long delay when the flowtable is full to the highest capacity. This is because the time to update entry is proportional to the number of entries in the flowtable. Therefore, more investigation is required to address the proactive flow entries consumption.

## 3. SDN Performance Challenges

Figure 5 presents the SDN performance challenges. This was derived from several combinations of search key terms for SDN, OpenFlow, flowtable, update, and security threats from various scientific academic research libraries, which include ACM Digital Library, IEEE Xplore, ScienceDirect, MDPI, Springer. Difference flexible search key terms were used in these websites to retrieve papers related to SDN performance challenges. The year of publication was filtered from 2015 to 2020, papers were chosen within the specified period as summarized in Table 1. After scanning, the selected papers totaling to 3905 are shown. It was further shortlisted and categorized into controller overhead, security vulnerabilities, switch flow table update operation, and flow entries management. Finally, a total of 88 papers were considered among all the research libraries websites.
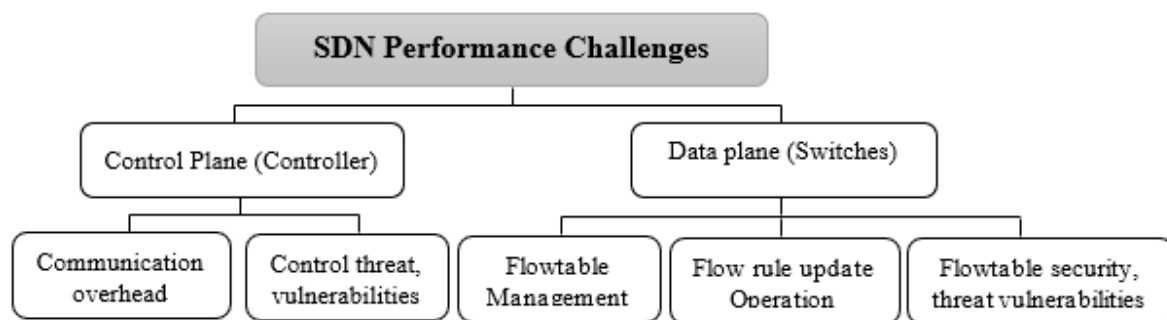


**Figure 5.** Taxonomy of software defined networking (SDN) performance Challenges.

**Table 1.** Number of article publications from research libraries.

| Academic Libraries | General Performance | Performance Challenges Related | | |
|---|---|---|---|---|
| | | **Controller Security, Threat and Overhead** | **Switch Flow Table** | **Others (Load Balancing, etc.)** |
| ACM | 1911 | 512 | 213 | 1186 |
| IEEE Xplore | 714 | 318 | 212 | 184 |
| ScienceDirect | 786 | 232 | 24 | 530 |
| MDPI | 317 | 69 | 68 | 180 |
| Springer | 177 | 23 | 26 | 128 |
| Total | 3905 | 1154 | 543 | 2208 |
| Considered papers | | 88 | | |

This work categorizes overhead into two; SDN controller and switches flowtable overhead. For the former in large-scale networks, the controller can easily be overloaded due to the large control message request from different switches, and thus cause high communication overhead and end to end packet processing delays [28]. While for the latter, potentially many rules are required to be installed in the switch flow table to efficiently operate; however, switch storage is limited, and as the number of flow rules increases, it overwhelms the switch storage memory [31]. Therefore, the following sections

explain four (4) major challenges: (1) communication overhead, (2) resource limitation, (3) rule update operation, (4) SDN security vulnerabilities.

### 3.1. Communication Overhead

DevoFlow [9] and DIFANE [12] are the state-of-the-art works to reduce the overhead of the controller by reducing the high volume of messages exchanged between the switch to the controller. DEVOFLOW aimed to reduce the communication overhead, switch—controller using fine-grained wildcards to improve the network-wide visibility of the controller. The mechanism leverages on rule cloning, thereby delegating some of the control function to switches to act locally in a way that the controller preserves the central network intelligence. DEVOFLOW modified the action field with CLONE flag, flag clear value signifies normal forwarding otherwise locally clone wildcard rule [32]. The cloning helps to process short live flows such as mice flow without the need to consult controller [3], hence, reduced the controller overhead. DIFANE presents a new architectural scalability solution built on the OpenFlow switch. The scheme maintains all traffic in the data plane by selectively distributing packets through intermediary switch called authority switches. These intermediary switches handle entry table-misses at edge switches, hence, reduce the need to invoke messages to controller. However, DIFANE and DEVOFLOW may require more complex software and hardware than conventional OpenFlow switches, which increases the cost of the device. Moreover, the fundamental principle of OpenFlow is to delegate all intelligence functions and control to the OpenFlow controller, therefore, such schemes violate this principle [33]. Conversely, Favaro et al. [32] introduced a blackhole mechanism to reduce switch to controller overhead while maintaining the benefit of the visibility for every new flow. Packet processing architecture is modified where only the first packet is forwarded to the controller while subsequent table-miss will be handled locally. In this way, the number of events sent to the controller is reduced. Although the use of blackhole has reduced controller consultation, it inherits weakness by its architectural design to drop notified packet, which may significantly result in a large number of packet losses.

As part of the effort to further reduce the workload on the controller and improve the efficiency of the data plane, the work of Kotani et al. [13] reduced controller overhead through packet filtering technique. The mechanism inspects header information aimed at reducing multiple packet-in messages forwarded to the controller. This is achieved by recording the information about packet-in messages forwarded to the controller. It filters duplicate information forwarded via packet-in while switches are to drop packets that bring overhead to the controller. The work of Avant-Guard [34] proposed control plane protection mainly to protect the controller from the TCP SYN flooding attack by an SYN cookies approach [13]. However, the work of [13,32,34] may also not cope with the nature of flows in a dynamic large-scale network.

### 3.2. Flow Rule Update Operation

TCAM is designed for high-speed packet lookup matching instead of fast flow entry updating time in the switch flow table. Rule updates are often required to adapt to the dynamic nature of network traffic patterns, especially in large-scale networks. During the update process, the controller instructs switches to add, modify, or delete some of the flow rules used to forward packets. For example, in Figure 6, the arrival of new flow in the switch flow table will require the firmware to examine whether it is the first entry in the flow table or not. The first entry will be installed straight away without further processing and priority order is maintained. For the case of subsequent entries, the switch firmware will get an appropriate position in the flow table, afterward, it will compare the priority with the existing entries, if the priority is higher, entry will be installed or entries will be moved down and comparison continued. In this way, the higher usage of TCAM, the greater number of entry movements. This method is referred to as priority base solution, which is the naïve solution to handle update operation and is widely used to implement OpenFlow switches. The process should be carried out safely and efficiently without service disruptions and network resources consumption (bandwidth and switch

memory). However, in dynamic and large-scale network traffic conditions, flow rule updates become more complex because of the need to re-order the existing flow entries [33]. These operations take some time to complete depending on the current memory status and are computationally difficult [35], regardless of the controller entries configuration mode being in either reactive or proactive mode [10]. An experiment shows the time to install a single rule after the occurrence of table miss, update cost from 5 milliseconds (ms) to 100 ms [29]. The time required to update TCAM is a non-linear function of the current number of rules. As reported in [10], adding 500 rules take about 6 s, however, the next 1500 rules take almost 2 min to install.

In contrast, to rule installation time, rule modification was reported to take a longer time because of the need to perform two operations in switches: removing the existing flow entry and installation of the new rule. Experimentally it was observed in [35] that modifying rules takes around 11 ms. Removal of the existing entry takes around 5 ms to 20 ms depending on the flow table usage. This incurs significant packet processing delay because of the need to store flow rules in descending order of physical address with priorities. It is important to carry out the rule update safely without having to tamper with an active flow rule. Removal of flow rule that corresponds to an active flow in the flow table forces switches to trigger packet-in message to the controller more often for flow setup request of that flow. This causes a significant processing load on the controller and may lead to a drop-in flow throughput besides an extra packet processing delay.
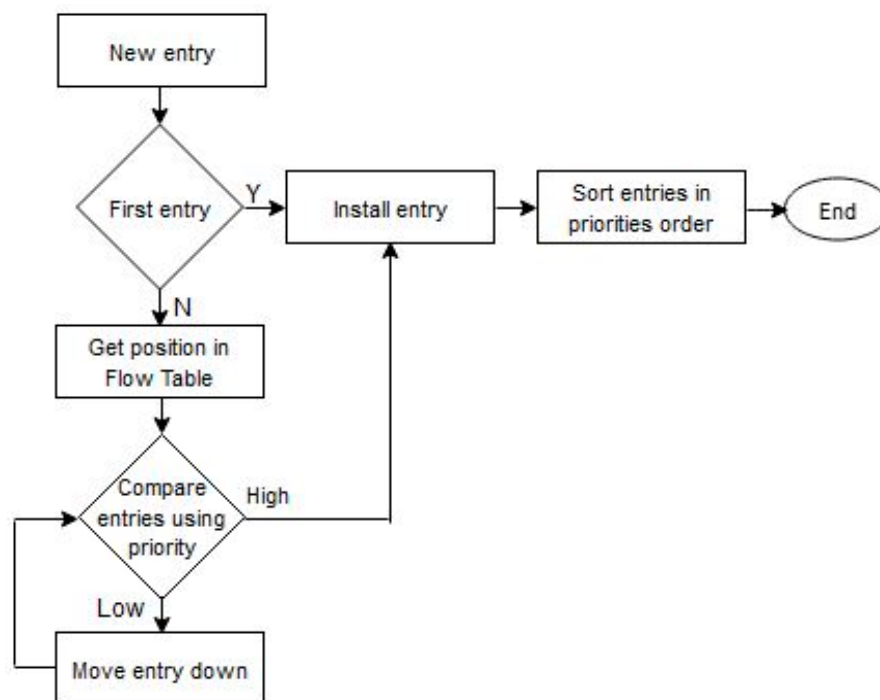


**Figure 6.** Example of TCAM Rule Update Operation.

Lately, the update operation concern has gained wider research interest. Some work has tried to address the issue by designing a scheme to minimize the number of flow entry updates sent to switch from controller [10,11,17,36]. For example, Update Cost aware [36], Balancer [10] can minimize the number of flow entry update by reducing redundant updates. Other proposals try to design a new firmware (ASIC) with an efficient algorithm such as FastRule [7,37], RuleTris [38] in switch ASIC that can reduce flow entry physical address movement in the TCAM. According to [37,38], a minimum dependency graph, like a directed acyclic graph (DAG), reported to have reduced the unnecessary flow entry movement update operation. However, applying DAG in ASIC required a policy compiler and TCAM update Scheduler, whose functions are to convert an entry update requirement into DAG and converting DAG into sequences of TCAM entry movement respectively [7]. RuleTris in [38] is the

state-of-the-art in designing an efficient policy compiler; however, it is computationally expensive in terms of TCAM update scheduler, which results in large firmware time up to 50 ms for a single update in flow table with a capacity of 1000 entries [7,37]. To overcome the shortcoming of [38], FastRule [7], RuleTris [37] reduces the flow entry update operation in terms of speed and time, 100× faster than existing solutions with a flowtable size of 1000 and operation time by 15 ms and 60% respectively.

The presence of a diversity of switch implementation is another factor that determines the performance of the flow update. There are a wide variety of switch implementations with different software control and hardware properties, which influence rule update operation efficiency. Most existing solutions overlook these differences and assumed all switches have the same hardware and software behavior. Zhoa et al. [18] noticed this problem and FastRule [7], RuleTris [37] are not an exception for such a concern, as neglecting these differences can lead to substantial performance challenges [18]. Diversity can cause significant utilization challenge, which can contribute to programming complexity. For instance, consider two or more switches of the same TCAM size, but one adds software flowtable as part of the switch. Adding the same sequence of rules may lead to rejection in one switch (TCAM full). Tango [39] noticed this problem and FLIP is an example for such a scenario [18]. This cannot be effective in practice. To address this problem while considering switch diversity, RuleTailor [18] proposed an optimization framework for the general TCAM-based switch rule update. RuleTailor aimed to reduce the rule update latency using techniques such as instruction type transformation, pseudo deletion, and match field distance, which effectively work for both control and data plane. Twelve Ms per rule update was achieved in the flow table with 1000 flow entries. In different efforts to further achieve less update, Balancer [10] achieved the least update cost by logically splitting TCAM into a reactive and proactive part with dynamic adjustment to efficiently replace unused rules. Although the schemes have made further efforts in reducing the update operations, it does not guarantee to maintain stable performance, especially with a large amount of network traffic flow. The dynamic nature of topological changes where the number of forwarding devices evolves rapidly over time, this can lead to significant performance bottleneck [3,40]. More investigation is required for adaptive flow entries update operation with least latency and overhead in a large-scale network environment.

### *3.3. SDN Security Threat and Vulnerabilities*

The noble idea of placing the network control logic at the central controller provides flexibility, optimizes network management, and flow monitoring, which is very important in the practical usage of SDN. Considering the SDN flexibility, it is expected that the potentialities of SDN could mitigate some security threats and vulnerabilities that were inherited in the traditional network. Unfortunately, SDN introduces another security threat and vulnerabilities due to its novel architecture [41]. At least five threat vectors were identified with SDN [41,42]. This includes attacks on vulnerabilities on SDN protocol [43], the central controller, underlying switches, forged legitimate traffic flows, and trust mechanism. Trust mechanism is very important in deploying application, overlooking it may easily cause malicious applications to be developed and deployed on the controller. It may further affect the detection of an event such as link failure in order to ensure fast recovery. In general, the most important threat vector is denial of service attack against the switch flowtable and central controller. For example, the central controller must have full control of the network state at a regular time interval to verify the network functionality and state [44]. In this case, the controller is required to receive an update from the switch more frequently. An attacker can leverage on the central entity to perform the DoS attack by forcing the switches to continuously update the controller with a fake update, which in turn consumed a lot of resources that the switch may use to process legitimate flows [44]. Similarly, because of the limited memory resources (TCAM), an attacker aggressively creates a large number of new traffic flows that might overwhelm the precious flowtable of the switches. In such a situation, the switches will be forced to continually add and delete flow entries, which may generate excessive packet-in message events to the controller. Consequently, the packet-in event will be stuck in the controller

queue as there would be no more routing decision taking place for every new incoming flow. In this case, flow with no corresponding entries will be stuck in the switches [45]. Concerning the protocol vulnerability, the widely used OpenFlow suffers from unexpected shortcomings during its design and evolution [43]. Discarding the transport layer security during the message exchange between controller and switches at the feature's verification stage was the greatest flaw. Man-in-the-middle attacks may leverage on such an operation to compromise the network. Similarly, exchanges of messages between switches lack the authentication/encryption mechanism in the SDN data plane [44]. Toward addressing the aforementioned security challenges, efforts were made lately to address several SDN security attacks [46–48].

The works of [42–45] proposed machine learning-based models to detect DoS attacks. Features were taken from SDN for data set under normal network state and DoS attack traffic. Afterwards, a new dataset was created using features selection methods on the existing dataset. This way, the model is trained both with and without the feature selection method under different classification models. The results have shown that machine learning and feature selection algorithms can detect DoS attack with high accuracy and less processing load on the controller. However, these solutions require keeping history as their performance can be acceptable when the flowtable space is quite enough. But in SDN where the flow table is a constraint with limited space, the methods may not give the desired performance.

Other works, including Shin et al. [49] and Kandoi et al. [50], analyzed the effect of DoS attacks on the network performance and reported how these attacks could affect many QoS parameters such as the switch to controller bandwidth consumption, latency, and switch flow table, controller efficiency. However, solutions to address these issues were not provided [45]. Towards this goal, Fung et al. [51] implemented FlowRanger as a controller application to mitigate DoS attacks. FlowRange consists of 3 components: (1) trust management (TM), (2) queuing management (QM), (3) message schedule (MS). TM is solely responsible for calculating trust value for each packet-in event based on its source, QM places a message in the priority queue corresponding to the queue value and MS processes messages according to weighted Round Robin strategy. To some extent, FlowRanger improved network performance and mitigated the impact of a DoS attack. This is achieved by prioritizing the serving of legitimate flow first in the controller. However, reference [45] noticed that FlowRanger does not prevent aggressive flooding the switch flowtable TCAM and controller. In contrast, Dridi et al. [45] leverage the OpenFlow features to the proposed SDN-Guard to protect SDN against DoS attack on controller processing capacity. SDN-Guard consists of three (3) component modules: (1) Monitoring module which monitors the traffic and decides where to redirect malicious traffic through the path with least utilized link in terms of bandwidth and switches TCAM. (2) Timeout management to assign timeout value to each rule according to probability. (3) Malicious flow rule aggregation, which assigns large hard timeout. SDN-Guard had optimized the network performance in terms of the parameters mentioned in [49,50]. However, generating a path using a monitoring module may not always be the shortest path, an increase in the number of traffic flow may augment the number of rules to be placed on the flow table, which in turn may increase the chance of flow table overflow. This issue was not peculiar to their initial work and is included in their follow-up work in [47]. It may be effective to conclude as the traffic flows increase, the scheme may introduce another loophole for DoS attack, which may force switches to refuse writing rules from a legitimate source. To mitigate such a problem, Luo et al. [46] proposed LRU flow entry rule eviction aimed to improve the SDN performance. The scheme can be effective with acceptable performance in fixed network with small flow table size, but in large network scale with dense flow arrival it may lead to significant communication overhead because of its architectural design of being reactive event-driven. Therefore, since DoS attacks are considered to be the biggest security challenges in SDN. Control and data planes always being the primary choice for attackers, more intelligent are required to reduce the interaction between the switch to the controller, furthermore switch flow table resources need to be efficiently managed, especially in large scale dynamic network environments to mitigate such concerns.

## 4. OpenFlow Flow Table Memory Management

As explained previously in Section 2.2, the switch flowtable is limited and cannot accommodate all required forwarding entries. Several proposals were made to improve the efficiency of the limited flowtable as illustrated in Figure 7. Some works proceed to manage flowtable rules by setting an adaptive timeout mechanism [52–55]. Other solutions consider reducing the size of the table based on the concept of aggregation. Some work focuses on splitting rules and distributes over the network in a way that satisfies policies according to the device capacity. Rule caching is another method to limit the number of concurrent forwarding rules in switch and reduce overhead switch to the controller. Machine learning techniques are quite effective to predict the traffic flow pattern and select the right flow to be installed in the switch flowtable. These proposals are detailed in the following sections.
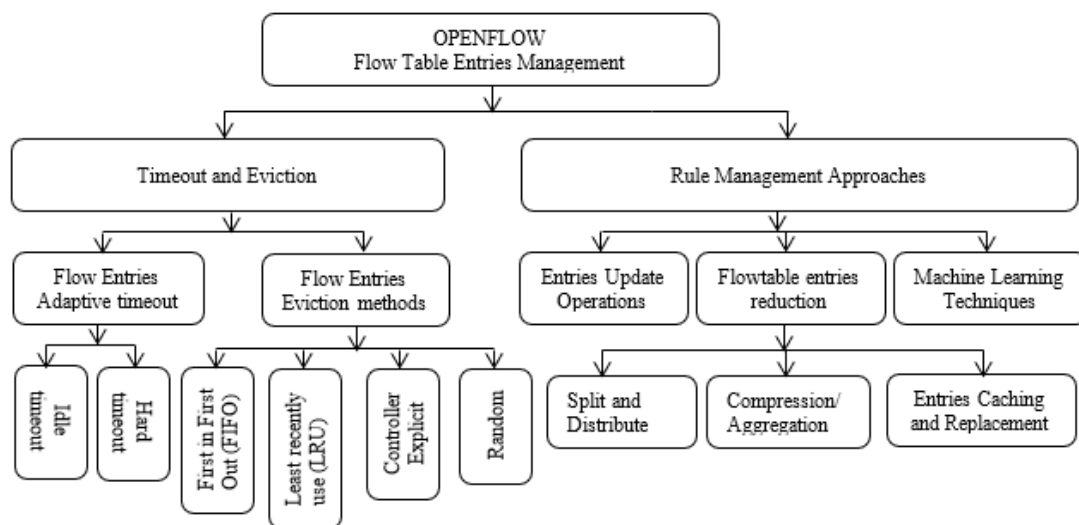


**Figure 7.** Taxonomy of flow entries management.

### 4.1. Timeout and Eviction Mechanisms

To overcome the limitation of the switch flow table storage (TCAM), OpenFlow allowed the use of a timeout mechanism to determine the life span of forwarding entry in the switch flow table [52]. When no packet matches an entry within its timeout period, it will cause an entry to be evicted from the flow table to free space for new incoming packets. Currently, there are two major methods for the OpenFlow controller to install the timeout mechanism, idle and hard timeout. OpenFlow controller usually configures flow entry with fixed idle timeout value in the scale of seconds across flows [56]. However, such timeout value could give better performance when it is close to packet inter-arrival time and the flow table has sufficient space to accommodate all flows. Moreover, this method could be straightforwardly implemented without much computational overhead, especially in a fixed network environment. However, setting fixed value across all flow may adapt to dynamic traffic flows especially in a large-scale network setting because of the variability exhibited by flow as illustrated in Figure 8. This has recently gained research interest to devise a scheme that will improve the flow lifespan, as shown in Table 2.

In Figure 8, suppose a number of packets that have been transmitted until time t1, t2 are vt1 and vt2, respectively. F1 represents 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. Therefore, it is required to investigate the feasibility of adopting a flow timeout strategy that can incorporate fixed and dynamic timeout values based on observed natures of intra-flow packet inter-arrival time in Figure 8 [46]. Applying fixed timeout value across the flow, which used to be the conventional method, makes it difficult to cope with the current traffic pattern, given that flows extreme exhibit variabilities in statistics such as duration

and volume. Moreover, [47] reports the impact of timeout value on flow statistics and found there is a performance trade-off. While large timeout value may unnecessarily preserve a large number of flows with no packet expected. As for small timeout may lead to premature eviction of flows, which increase flow set up request and teardown long live flow into a smaller flow. Consequence flows may be evicted more often, which causes the generation of a more packet-in message to the controller, such a situation increases communication overhead between switch to controller.
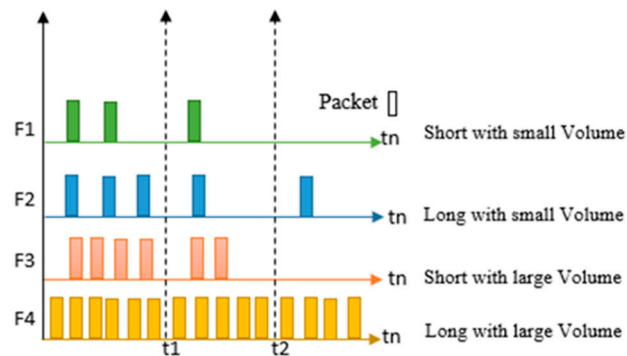


**Figure 8.** Example of traffic flow with variability of inter arrival time.

**Table 2.** Comparison of proposed flow timeout mechanism.

| Related Work | Controller Placement Mode | Method | Rule Eviction | Timeout Mode |
|---|---|---|---|---|
| Lu et al. [47] | Reactive | Traffic feature-based idle timeout lognormal distribution | X | Idle |
| Challa et al. [48] | Proactive | Bloom filter | Data logging using multiple bloom filters (MBF) | Idle |
| Xu et al. [49] | Reactive | Combine flow table and controller (CFC) | Idle timeout eviction | Idle |
| Li et al. [50] | Reactive | Flow table adaptive timeout algorithm | X | Idle |
| Liu et al. [51] | Reactive | Lognormal distribution using probability | Random | Idle |
| Kim et al. [20] | Proactive | Flowtable vacancy and mathematical model | Least recently used (LRU) | Idle |
| Guo et al. [52] | Reactive | Software defines adaptive routing (STAR) | LRU | Idle |
| Panda et al. [57] | Reactive | Dynamic hard timeout allocation | LRU | Hard timeout |
| Huang et al. [53] | Proactive | Timeout calculation: idle timeout with two stage-table | Controller randomly evict flow with no match | Idle |
| TimeoutX [46] | Reactive | Composed of 3 modules: history flow information base (HFIB), timeout selection algorithm (TSA), and EIMC | Entry installation and management component (EIMC) | hard |
| IHTA [54] | Hybrid (reactive and proactive) | Dynamic idle and hard timeout based on traffic pattern to reduce overhead | Based on flow packet count | Idle and hard timeout |

Lu et al. [47] noted the factors that influence setting a proper timeout such as packet inter-arrival time (PIAT) according to flow state, controller capacity, and current flowable utilization. Importantly, setting proper idle timeout should be based on the packet inter-arrival time and can effectively improve the efficiency of the system. In this way, the work of Lu et al. [47] set a TF-idle timeout scheme that dynamically set the flow entry lifecycle according to real-time network traffic. This is achieved by estimating traffic feature via packet arrival distribution and interval in a logarithmic normal distribution. Since it is observed that general networks change over time with variability in flow size. The method required collecting statistical distribution of packet arrival time. Consequently, the cost of memory in the controller is huge. Similarly, flow removal should follow a certain procedure to prevent evicting important flows. TF-idle only focuses on setting timeout without considering flow eviction. It is important to also consider the flow eviction together with the timeout method. This is necessary to avoid removing an entry that may be needed in the near future. Alternatively, instead of relying entirely on the flow expiry period, Challa et al. [48] proposed to efficiently manage the flow entries in the flow table through intelligent eviction mechanism based on data logging using multiple bloom filters (MBF) data structure to decide flow entry removal. Although the approach made a step further to remove the unused entries. However, the processing of encoding some important value of flows in SRAM incurred an extra processing delay to the system and the worst case is that flow traffic pattern variability may impair the stability of the system.

The work in [49] proposed an adaptive flow table adjustment by combining flow table and controller cost according to the proportion of active flow entries. The algorithm monitors traffic in real-time. Hence, the algorithm dynamically sets the value of idle timeout based on different flow. Interestingly, the algorithm could set the different timeout values to flows. However, the process of calculating the cost of flow table entries in switch and controller computing cost, add an extra computational overhead on the controller. Consequently, it may limit the number of switches that could be handled by the controller [55]. Moreover, obtaining the real-time flow entries through (*of.ofp_stats_request*) message to the switch before the flow finish is impractical [46,58].

In a similar effort, [50] modeled flow as ON or OFF to analyze the packet length and packet interval length was used to assign suitable idle timeout. This required a switch to perform some calculation upon arrival of a packet. Similarly, the controller maintains a history of flows to adjust the idle timeout value dynamically. Even though the approach may be stable with a proper idle timeout when the flow arrival is in a uniform distribution; however, change of flow shape to exponential distribution may significantly introduce overhead to the switch. This is due to the process of switch calculating the flow table resource overhead upon arrival of a packet that may not be feasible in real-time [46]. In addition, it may reduce the number of flows to be processed by switch at sampling time. Moreover, a packet may incur an extra processing delay. Their work overlooks incorporating flow eviction methods.

The work in [51] introduced forward a dynamic adaptive timeout algorithm to install different timeout values at the sampling time period. This is achieved by estimating the number of flow entries that may appear at the next sampling time in probability. Thereafter, an idle timeout is chosen according to the estimation. To cope with the dynamic nature of traffic, the controller used to change the timeout value at every sampling time. Although the algorithm may be efficient with low flow arrival as the number of flows increases, this method fails to give a good performance, because it significantly imposes heavier pressure on the controller with a high cost of computing power [59]. In addition, the approach uses random policy to remove the unused entry when the timeout expired. Randomly evicting flows may cause an active flow to be removed, which can severely degrade the performance of the network with extra processing load on the controller [52].

The work of [52] designed a module operated at the switches to enable effective flow table entry management to remove entries using LRU and timeout to ease the impact of flow table overflowed. A counter is set at the switch to count the number of active entries, but it is noted that the method may lead to miscounting when there are duplicate in SYN/FIN packets. As such, to avoid miscounting, every entry is associated with a binary flag to determine the states of entry, 0 and 1, which indicate

inactive and active states, respectively. Therefore, the method may evict unused entries intelligently when needed to accommodate new flows; however, it requires modification of the OpenFlow switch flow table data structure, which makes it a computationally prohibitive solution. In addition, when the timeout of entry elapsed, LRU is used to evict an entry and the use of an LRU in a flow-based network is not appropriate. Because this algorithm is based on single rule replacement for each switch, whereas proper control policies should be bases on a global view of multiple rules in all switches [60].

Similarly to preserved useful entries in the flow table, the proposal in [53] designed a flow cache architecture with a two-stage timeout mechanism to better identify and keep important flow entries. Initially, flow entries are stored in the primary table with timeout calculated by the controller using knowledge of flow entry. Thereafter, an inactive time with no packet expected, instead of completely removing entries, but are moved to a secondary table for a second chance. It is noted that the approach preserved the active flows, thereby evicting short-lived flows to avoid waste of flowtable resources. However, it required knowledge of flow before deciding on a suitable timeout. Hence, it is not feasible to obtain old flow installation, evict, install, and packet-in in real-time [46]. In addition, moving an entry to secondary storage for the second chance, the process may cause a packet to incur additional processing delay, which may not be good for delay-sensitive application [55]. Therefore, setting proper idle timeout is significant to efficiently manage the precious flow table space. Additionally, it is also important to consider the deterministic life span of the flow entry (hard timeout) to preserve long-duration flows with a large number of packets within a short inter-arrival.

In contrast to the idle timeout features, Panda et al. in [57] consider a dynamic allocation of hard timeout for predictable and unpredictable flows to maximize the utilization of flowtable. The aim is to keep an unpredictable flow for a short time while preserving predictable flows. This is achieved by analyzing packet trace to study the nature of packet arrival for each, thereafter adjusting hard timeout accordingly. When flow table utilization reaches a certain capacity, LRU is used to evict an entry with a max timeout. Even though the scheme has achieved in preserving predictable flows, still inherent is the problem of poor entry eviction scheme. LRU is not an appropriate eviction algorithm in SDN because it is a packet driven algorithm, which can only be implemented in theory, but in practice it may not be compatible with OpenFlow, and worse, it violates one of the SDN principles that deleted all control functions to the controller [33].

IHTA in [54] considers the dynamic idle value to short live flows and adjusts hard timeout value to long live flows with short packet inter-arrival time. This has significantly reduced the issuance of the packet-in event to the controller, which improves the communication overhead. The shortcoming of LRU is overcome by leveraging on OpenFlow built-in data collection. Flows with a small number of packet count are considered to be victims and therefore, removing such flows improved the efficiency of the limited storage. It is effective to conclude that to some extent IHTA has reduced communication overhead and improved the limited flowtable usage without changing the architecture of SDN.

*4.2. Flow Rule Aggregation*

SDN has been designed to support a wide range of network applications that are flow-based with more complex matching patterns compared to destination base using traditional IP routers [56]. The complexity comes as a result of forwarding rules stored in TCAM, which is a constraint in size, and represents an important concern in deploying OpenFlow technology. One of the alternative solutions in reducing the demands of TCAM is flow entries aggregation, a technique that reduces the number of required entries by merging multiple flow entries into one aiming to preserve the original forwarding semantic through wildcard rules. Table 3 summarize the related works.

Intuitively, the technique compresses fine-grained forwarding entries into fewer coarse-grained with slightly larger matching range, thus a reduced number of entries to be stored. Interestingly, it is a software solution that is easy to implement as an optional plug-in on the OpenFlow controller with no modification to OpenFlow switches or protocol [61]. There exists aggregation techniques for traditional prefix IP routing table or non-prefix TCAM forwarding rules or farewell rules (ACL) [59,62,63]. However,

the aggregation technique poses some challenges when updating forwarding entries or querying traffic statistical counter by the controller, because of the failure in preserving the original semantics of the rule in most cases. In addition, because of the diversity of switch vendor implementation, some OpenFlow switches may not fully support wildcard values matching fields [59]. Moreover, a lot of events such as node failure, QoS provisioning may cause frequent flowtable rule updates in practice. As such, an aggregation scheme must be efficient enough to adapt to the nature of frequent rule updates with less computation and fast update time. To this end, Tsai et al. in [64] and Luo et al. [65] used bit and subset weaving and fast flow table aggregation (FFTA), respectively, to shrink the flow table size and provide practical fast updates time. The FFTA construct optimal routing table constructor (ORTC)—based aggregation with binary search tree (BST) to optimally aggregate rules with bit merging on the tree. The scheme leverages on bit weaving and removes bit swapping to construct BST of non-prefix flow entries to improve the execution time. It was observed that some rules are unaggregatable, therefore FFTA misses the potential opportunity of aggregating such flow entries [66]. Moreover, as the size of the network increase, it will be computationally expensive.

**Table 3.** Comparison of the related works of flow table entries reduction mechanism.

| Related Work | Controller Placement Mode | Method | Goal | Use Case |
|---|---|---|---|---|
| Cheng et al. [67] | Reactive | Quine-Mcclustkey algorithm. Hidden Markov model (HMM) | To manage the multiple flow table and reduce flow processing time | Flowtable management |
| FFTA [65] | Reactive | Shrink the flow table size using binary tree aggregation | Reduce flowtable size with fast rule updating time using aggregation technique | Flowtable management |
| FTRS in [61] | Reactive | Rule optimization and binary trie aggregation | To reduce the number of flow entries needed in the almost full-filled flow tables, while retain the original QoS | Flowtable management |
| IDFA [66] | Reactive | Redundant flow entries with dynamic threshold value aggregation | Reduce flowtable overflow problem and flow aggregation convergence time. | Flowtable management |
| Kanan et al. in [68] | Reactive | Flow entry compression | Reduce flow table size by compressing matching header | Flowtable management |
| OBS [69] | Proactive | One big switch | Distribute rules via abstracted forwarding element called "one big switch" | Distributed ACL and load balancer |
| Minnie [56] | Reactive | Flow entry compression | To maximize the utilization of SDN switches flowtable | Traffic engineering |
| Tsai et al. [64] | Reactive | Bit and subset weaving to merge flow entries to a subset of a partition | Reduce flowtable size with fast rule updating time using aggregation technique | Flowtable management |
| Palette [70] | Proactive | Flow entries split and distribute | Decompose flow entries into smaller part and distribute them across forwarding element | Distributed ACLs |
| OFFICER [71] | Proactive | Linear optimization model | Modeled rule allocation problem in resource-constrained OpenFlow networks with relaxing routing policy | Traffic engineering |
| Sheu et al. [72] | Proactive | Break tables into a number of smaller sub-tables and distributes them across network switches. | Ternary content addressable memory (TCAM) shortage problem through distributing rules | Distributed ACLs |

FTRS [61] is another flow table reduction scheme. FTRS preserves the existence of each entry at the edge of the flow path and aggregates the flow entries that have the same action that is destined to the same destination address. To achieve this, FTRS identified core switches that are more likely to encounter flow table congestion than edge switches. In this way, less important flow entries are aggregated in the middle of the flow path while preserving the existence of entries at edge switch to maintain fine-grained management. Binary tree (prefix tree) is used to traverse a selected matching attribute from each flow entry such as IP address into node, thereafter, flow entries are reduced by

replacing the non-empty trees with coarse-grained nodes. To some extent, FTRS has reduced flowtable consumption, but it fails to aggregate some types of flows.

To address the shortcoming of FTRS, Chao et al. [66] proposed an in-switch dynamic flow aggregation (IDFA) mechanism as a switch application to avoid the link between the switch to the controller. IDFA adds redundant flow entries insertion and verification to dynamically adjust and aggregate entries based on a threshold value. IDFA claimed to have better aggregation convergence time with a lower chance of overflow compared to FTRS. However, its performance may be acceptable in a fixed network, but it may introduce extra overhead, especially in large-scale networks with heavy-hitting flows. Besides, IDFA and FTRS are more applicable to only a small local area network (LAN) environment [67].

Alternatively, OpenFlow's 1.3 support multiple flow tables (MFT) pipelining stage to offer entries matching flexibility. As such, the work of Cheng et al. [67] put the Agg-ExTable approach to efficiently manage entries in MFT. The approach periodically aggregates flow entries using pruning and Quine-Mccluskey algorithm. Hidden Markov model (HMM) is used to determine the most popular entries with heavy matching in probability, such entries are placed in the front-end ExTable, while others are placed in the subsequent table. Experimentally, the scheme saves about 45% memory space and reduces the flow processing time. However, the weakness of Agg-Extable relies in its inherent architecture, as this solution required large number of entries in the front-end table, which may significantly pressurize the ExTable and can lead to overflow problem. In addition, it may be challenging to adopt such solutions in large scale networks, especially with the variability exhibited by flows.

Kanan et al. in [68] compact TCAM entries aimed to reduce the size of the flow entries in the flow table by inserting short tags in the packet header to identify the original number of bits used to store entries for OpenFlow switches. This is achieved by leveraging the dynamic programming capability of SDN to route the packets using those short tags. The approaches have optimized the TCAM storage space with a significant reduction in TCAM power consumption for a given number of flows. However, the solution required a change in the packet header and method to populate the flow table. Moreover, adding an identifier to each incoming packet in ASIC-based storage is hard, because it is not the standard operation, thus, causing the packet to be processed slowly with the performance penalty [56]. Rifai et al. in [56] proposed Minnie, which composed of two modules; compression and routing module. The flow table is configured with 1000 rules as the size when the number of rules exceeds the 1000 limit, Minnie triggers the compression module, this way rules are compressed by source, destination, and default, an optimal rule is chosen as the best rule. Thereafter, the shortest path is used to route the flow without overloading the flowtable on the topology where the number of nodes increases. Minnie focuses on achieving better compression ratio and computation time; however, with the variability exhibit by flows, some flow tends to live for a short or long period. Minnie overlooks incorporating flexible timeout and eviction mechanism to remove useless entries to free space for a new flow. Besides, the different compression methods employed may introduce another overhead with system instability.

### 4.3. Flow Rule Split and Distribute

Endpoint policies like firewall, load balancer rules are enforced at edge switches. These policies always generate many forwarding rules. Due to insufficient flow table space (TCAM), a set of rules are usually split into small sub-tables and distributed over the network in a way that satisfies policies. The common objective of distributing schemes is to minimize the number of entry rules for realizing policies in each switch. This way, optimization models are presented to decides which rules to be placed on which switch while respecting memory constraint and rule dependency. Palette in [70] is an example of such a scheme, Palette designed a scheme to share rules to individual ingress switch, thereafter, distribute across the network, such switches in the network have smaller sub-tables. They formulate two greedy approaches using a rainbow path coloring problem to optimize the entries in the sub-table. However, balancing the size of each sub-table and reduce rule redundancy concern is noted as the

shortcoming of the solution [72]. To address such challenges, Shue et al. [72] present a sub-table allocation algorithm (SA) and size-balancing sub-table partition (SSP) algorithm. This way, the idea of coloring is extended to maximize the number of sub-tables (i.e., number of colors). SA algorithm can partition a large number of sub-tables to reduce table size. Experimentally optimal result were achieved in small network topology, although, compared to Palette [70], SSP can balance all the sub-table sizes with low rule overhead. Moreover, SA plus SSP may occupy little TCAM space to store rules regardless of the topology size. However, the arrival of massive flows in a dynamic large-scale network may impair the stability of the system.

Kang et al. [73] formalize a split model as decomposition on overlapping rectangles with linear programming (LP) to assign sub rectangles to switches. However, it introduces unwanted traffic because policy enforcement is executed at several hosts after packets entered the network [74], moreover, it may not be possible for distribution to ensure feasible space allocation for each switch in the network. In a similar effort, one big switch (OBS) in [69] considered rule distribution problem under heterogeneous flow rule policy. The author introduces the LP base rule partition approach with two heuristics for rule allocation to calculate suitable rule space allocations for each flow path in each switch and therefore only the rules that affect packets traversing are installed. However, OBS represents endpoint policy rules by five dimensions, the computation cost of table decomposition is impractical [72]. The work of Lin et al. [74] presents table decomposition and sub table allocation for heterogeneous flow table with efficient rule distribution method to balance the number of rules stored along with a path. However, the method may cause incorrect packet forwarding due to the rule dependency problem [72].

Conversely, Nguyen et al. [71] proposed OFFICER as a linear optimization model for sub-table allocation under memory and link capacity constraint aimed to calculate and implement efficient forwarding rules in switch memory. OFFICER treats the network as Blackbox, which must satisfy the endpoint policy imposed by the operator, thereby obtaining the maximum from available resources through the adaptation of routes. Therefore, in general, splitting and distributing entries among data forwarding elements can maximize entries. However, it introduces more traffic overhead in the switches to maintain the distribution of forwarding entries, especially when the network topology dynamically changes over time. Moreover, the distribution scheme may be a bottleneck in large scale-network with massive flow arrivals.

## 4.4. Flow Rule Caching

The use of wildcard rule caching makes multiple flows aggregately, thereby reusing entries among different flows, communication overhead can be drastically reduced. In addition, the cost of policy enforcement and update can be reduced. However, wildcard rules are usually stored with different priorities assigned to resolve conflict between two or more overlapped packets. For example, let $p$ represents a set of rules R (R1, R2, R3 . . . RN) N-dimensional rules with the following priority order (R1 > R2 > R3 > . . . RN). There exists rule dependency if the match field of rule $R_1$ overlap at the intersection field of rule $R_2$, and rule $R_2$ priority is higher than $R_1$. Therefore, caching R2 only due to its high matching frequency without caching R1 may result in incorrect packet forwarding. Hence, it is not enough to simply cache the requested rule when there is no available correspondent rule, all other rules with higher priority must be installed to guarantee correct packet forwarding. In such a situation, an extra storage overhead is required, which increases flow table overflow [75]. This has widely gained researchers' interest, as summarize in Table 4.

Alternatively, it is to rely on additional data paths to process the remaining traffic flow. The recent advancement of technology introduced software switching as part of the same commodity switch, which provides an alternative by storing a large number of rules. To some extent software switch has reduced the storage limitation problem; however, it does not support wildcard rules that match many headers fields. In this situation, the switch must resort to slow processing in user space to handle the first packet of each flow. This causes significant packet processing delay, which may not be suitable for some applications. Thus, based on the nature of the amount of traffic matching switch rule,

Katta et al. [11] suggest another option to combine hardware and software (hybrid) switching to get the benefit of both without having to pay for their drawback. But this requires careful distribution of single rule table into multiple tables spanning across heterogeneous data paths to ensure that semantic of single-switch rule table are preserved in the distributed implementation. To some extent, this can offer large storage space to accommodate more traffic. However, the process of forwarding packet to software switch thereby copying it back to TCAMs flow table incurred additional packet processing delay. Such an additional delay is high, which amounts to at least 20 ms, Mohan et al. [76]. Subsequently, as the data flow request increases over time, the time augments.

In contrast to hybrid switch, other solutions were proposed using commodity switches with no additional resources to use TCAM only to store rules. Rules are partitioned into two; heavy-hitting rules and others. the first partition stored heavy-hitting rules while the remaining rules are placed in the other partition [77]. Therefore, partition in TCAM only nullifies the additional packet processing delays experienced in the hybrid switch, however, the potential rule dependencies caused by wildcards to complicate the installation and removal of the rule. It is very common for higher priority rules to overlap at the intersection field with lower priority rules. When packet matched with multiple overlap rules, the higher priority would be the final match. As a result, a problem of rule dependencies issues arises [31]. Therefore, simply caching the requested rule could potentially lead to incorrect packet forwarding. In such a situation, an extra storage overhead is required, which increases flow table overflow [78]. Moreover, this will block the chance of other new incoming rule and cache miss will increase as well. Consequently, this situation will further increase the packet-in generation and update cost operation. An alternative method is to convert wildcard rules into new micro/exact rules without overlapping [75]. However, each rule has tens of overlapping rules, slicing the long dependencies chain into non-overlap rules generates a large number of flow entries [78]. This would eventually overwhelm the switch flow table (TCAM) memory. In this way, the extra processing load on the controller to store and update rules is inevitable. Therefore, it is necessary to carefully handle dependency in order to avoid overwhelming the switch flow table memory and avert incorrect packet forwarding.

**Table 4.** Comparison of the related works of rule caching mechanism to improve flow table utilization.

| Related Work | Controller Placement Mode | Method | Goal | Use Case |
|---|---|---|---|---|
| Katta et al. [11,79] | Proactive | CacheFlow: cover-set using Direct acyclic graph (DAG) | Improving the efficiency of TCAM. Allocate rules b/w TCAM & RAM to solve the rule dependency problem | Distributed ACLs |
| Sheu et al. [73] | Proactive | k-Hop neighbouring Set | Improved cover-set to solve rule dependency the problem | Distributed ACLs |
| Ding et al. [36] | Reactive | Uses DAG to solve the replacement problem | TCAM replacement problem under rule dependency constraints | Distributed ACLs |
| CUCA [31] | Mixed mode (reactive and proactive) | Mixed cover-set and partition | Allocate rules b/w TCAM & RAM due to rule dependency problem | Distributed ACLs |
| CAB [75,78] | Reactive | Partition with buckets using a decision tree | To mitigate the dependency problem by partitioning the field space into buckets and caching rules associated with the requested buckets. | Distributed ACLs |
| Wu et al. [80] | Proactive | Forest tree to install a branch of rules using dynamic programming (DP) method | To maximize the number of rule hits, while limiting the number of cached rules. | Distributed ACLs |
| Wang et al. [81] | Reactive | Decision tree to install a chunk of rule | Intelligent rule management scheme to reduce communication overhead | Distributed ACLs |
| Wang et al. [34] | Reactive and proactive | Hybrid timeout | To handle rule dependencies with flexibly hybrid timeout mechanism | Distributed ACLs |
| CRAFT [77] | Proactive | Two-stage caching architecture called CRAFT for the flow table | To solve rule dependency problem using two-stage cache | Distributed ACLs |
| IRCR [82] | Reactive | In-switch rule caching and replacement (IRCR) replaces a rule according to the expected number of incoming matched flows | To reduce flowtable overflow problem | TCAM flowtable management |

Rule caching algorithms cannot blindly choose and cache the most frequently used rules. Hence, a scheme must be properly designed to choose the most frequently used rule and cache them in the TCAM flow table. Toward this goal, cover-set (CS) was proposed, and it is built based on the hybrid switch. CacheFlow in [11,73] was the state of art work to design CS, it leveraged on direct acyclic graph (DAG) to represent the wildcard rule dependencies problem by splicing a long dependency chain. CS algorithm creates a small number of new rules that cover many low priority rules that overlapped with other higher priority rules. Rule weight is used to indicate how frequent the rule is being matched. Rules with higher weights are considered as heavy-hitting rules, therefore, such rules are cached in the TCAM flow table while the remaining sets of lower priority rules are forwarded to software switch. Sheu [73] noted that CacheFlow only considered a contribution value of each un-cached rule to select the heaviest hitting rules.

In contrast, Sheu [73] proposed accumulated contribution value (ACV) for a set of rules to further improve the selection of most frequently used rules. The set of related rules that have maximum ACV are cached in TCAM until full. The work in [31] is another example of the CS algorithm. To some extent, CS can accommodate a large number of flow rules, but the merit comes with a cost. CS does not improve the efficiency of TCAM, instead it may increase the chance of flow table overflow as the number of long chain dependencies increase because of the large number of cover-set rules that must be stored in TCAM [77]. In addition, the process of forwarding packet to software switch, thereby copying it back to TCAMs flow table, incurred additional packet processing delay. In general, the weakness of cover-set and similarly [31,72,73] relies on their inherent architectures, as these solutions require the installation of software switch for every hardware switch [56] that might need a reorganization of the network cabling and additional resources to host the software switches [59]. In addition, it is difficult to determine the optimal number of needed software switches due to performance reasons. Moreover, forwarding rules that depend on the traffic characteristics must be kept in software switch kernel memory space, which is also limited. Furthermore, the process of software switches over a hardware switch increases packet processing delay to consult the controller and install the missing rules [77]. To this end, a greater number of update operations will be triggered, which also affect the stability of the systems.

In contrast to cover-set, partition mechanism in the work of [75,77,78], caches flow rules in TCAM without using additional resources. It divides large dependent rules into sub-set and places the most frequent rules in the first partition and less frequent one in the second partition. CAB is the state-of-the-art wildcard rule caching using a partition. Yan et al. [78] used a two-stage flow table pipelining to address the rule dependency problem. The main idea is to divide the flow rule into many small logical storages (buckets) and use a decision tree to partition rules. The design tries to ensure one bucket never overlaps with other buckets, but it is possible for rules to overlap with multiple rules in different buckets. In the first stage, bucket is cached while in the second stage all associated rules are stored. CAB can reduce packet processing delay with a better cache hit ratio in the fixed network and suggest improving the scheme to cope with dynamic traffic patterns as future work. The work of Li et al. in [77] extends the concept of CAB to introduced CRAFT. This scheme used two-stage pipelining to reduce the packet processing delay experienced in CacheFlow [11]. Base on experimental results CRAFT outperforms CacheFlow in terms of hit ratio on average by 30%.

Similarly, [81] uses a decision tree to partition rules as a chunk and reactively installed it at the switch flow table. When chunks become inactive, it is uninstalled with its associate. Therefore, wildcard rule caching using partition has improved the TCAM flow table efficiency compared to cover-set as observed in [31,77]. However, from the work of [75,77,83], regularly partition rules led to many sub-rule creations, which will consume a lot of switch memory space. Conversely, maintaining rules in their original shape without splitting them to prevent fragmentation generates a large number of buckets [31]. In this situation, a single rule may be duplicated in several buckets and therefore a lot of spaces would be wasted.

### 4.5. Machine Learning Techniques

Predicting flow traffic patterns is crucial to select the right flow to be installed in the switch storage to reduce the large flow entries, which lead to inefficient utilization of the network resource. The main idea is to provide a way to perform fine-grained network management by identifying different flow types due to variability exhibit by flows. Therefore, flow classification assists network operators to handle the better quality of service (QoS) provisioning and resource utilization in a more efficient way. Network QoS is simply the service level agreed upon to deliver network application to users. It is usually measured through end to end delay, bandwidth, jitter, and packet loss. There is a need to efficiently facilitate measuring and configuring such services to meet the users demand. OpenFlow has built-in data collection modules that can collect statistics of switches, ports, and flows, which are used to facilitate flow monitoring to maintain the global network knowledge [84,85]. However, other important metrics such as port utilization, delay, and jitters are not directly available [86]. Port utilization can be measured by using some statistical method, whereas delay and jitter require an extra feature to be measured. Therefore, to efficiently preserve QoS provisioning, more intelligence is needed to be deployed. In this way, machine learning (ML) techniques are applied to extract knowledge from the statistical traffic flow data gathered by the controller. Thus, leveraging the global network state to apply ML algorithms in the realm of SDN from the perspective of routing optimization [87], resource management [83,88], QoS quality of experiences, and traffic classification prediction [82,89] have gained a lot of researchers' interest lately, refer Table 5.

Traffic flows can be classified into different QoS categories, whereas prediction techniques are used to forecast the next traffic expected. To flexibly ensure QoS as well as the quality of experience (QoE), deep packet inspection (DPI) is very effective and widely used for traffic flow classification due to its high classification accuracy [3,85]. However, DPI cannot recognize an application whose pattern is not available, besides it has high computation resources cost because of the need to check all traffics flows. Moreover, it cannot classify encrypted traffic on the internet [90]. Applying such techniques in large scale networks with exponential traffic growth make pattern update difficult or even impractical [85,90]. Conversely, ML-based approaches are more effective in recognizing encrypted traffic with much lower CPU computation cost compared to a DPI-based approach [3,91]. Although ML has lower accuracy, it is still effective in SDN compared to the legacy network due to network global and intelligence present at the SDN controller [3,90]. Hence, researchers leverage the controller intelligence to implement the ML technique. Works have been conducted to classify traffic flows from a different perspective to adaptively provision QoS and efficiently manage the precious storage resource. Such classification includes application-aware and elephant-mice flow aware.

The former focuses on identifying different delay-sensitive and non-sensitive application traffic flows. Such an application required immediate detection and redistribution along the network to minimize declining SDN QoS policy. However, with the rapid increase of applications on the internet, it would be impractical to identify all the applications, especially in a large-scale network. The work of Amaral et al. in [82] developed an OpenFlow-based SDN system to collect statistical data and classify network traffic flow in an enterprise network. Afterwards, classifier algorithms are applied to classify traffic flow into different application categories. Similarly, Rossi et al. [89] aimed at devising a behavioral classification engine to give accurate application-aware traffic while focusing on user datagram protocol (UDP).

Concerning the latter, it focuses on identifying elephant flow, whose lived-long with high bandwidth consumption and mice flows that live-short, which are delay-nontolerant flows. Since elephant flows have high volume in nature, they have a high tendency to fill the flowtable storage. Such classification is necessary, which can be used by network operators to optimize the usage of the limited network resources according to their desired QoS. Glick et al. [92] put an ML-based technique to study flows based on elephant flow-aware at the edge of the network. Thereafter, the controller utilizes the classification results to implement efficient traffic flow optimization algorithms. Xiao et al. [93] focused on a cost-sensitive learning method to detect elephant flows. The proposed method comprised

of two stages: In the former stage focus on head packet measurement to different mice and elephant flows, while the latter leverages on a decision tree to detect and analyze whether these flows are genuinely elephant flows or otherwise. Although, SDN approaches can efficiently allocate resources to different flows the overhead for storing a large volume of flow entries is significant, especially with the memory scarce.

**Table 5.** Comparison of machine learning related work to improve flowtable storage.

| Related Work | Controller Placement Mode | Focus | Technique | Use Case |
|---|---|---|---|---|
| Sminesh et al. [77] | Proactive | Flow monitoring to reduce congestion and packet loss in SDN | Experimental validation | Routing rules |
| Amaral et al. [82] | Proactive | Traffic classification | Supervised learning | Routing rules |
| Rossi et al. [89] | | UDP flow classification | Support vector machine supervised learning method | Routing rules |
| Glick et al. [92] | | Elephant flow-aware traffic classification at the edge of the network | Machine learning technique | Traffic flow routing rules in DC. |
| Xiao et al. [93] | Proactive | Learning method to detect elephant flows in SDN | Decision tree | Routing rules |
| Yang et al. [94] | Proactive | Predict the duration of the flow entry | Machine learning technique | Routing and distributed ACL rules |
| FlowMaster [95] | Proactive | Predict when flow entry becomes stale | Probability | Routing rules |
| Al-Fuqaha et al. [83] | Proactive | Machine learning techniques to decide the preserved flow between long-lived (elephant) and short-lived (mice) | Deep learning neural network | Routing rules |
| Yang et al. [88] | Proactive | Classify flows into active and inactive to decide the right flow to remove intelligently | Machine learning techniques | Routing rules |
| Li et al. [96] | Proactive | Q, learning approach to an efficiently select flow timeout value | Machine learning techniques | Distributed ACL rules |

Therefore, it is crucial to determine which flows between short and long live flows with a massive number of packets will be preserved in the flowtable and which flow entries should be processed by the controller. This is required to satisfy the goal of which flow between elephant and mice can reduce both storage and controller overhead while maintaining stable performance. Toward this goal, the work of Al-Fuqaha et al. [83] proposed an ML-based system that leveraged on two variations of reinforcement learning (RL) with traditional RL-based algorithm and other deep RL to determine the right flow to preserve the flowtable between elephant (long-lived) and mice (short-lived) using deep learning neural network. Yang et al. [94] proposed machine learning-based systmes to decide the right flow to be deleted. The approach learned from the historical data of flow entries afterward, it predicts how long entry can last, flow entry with the shortest duration will be the victim. In a similar effort, FlowMaster [95] predicts when a flow entry becomes stale in order to delete such entry to free space for next incoming flow. However, FlowMaster assumed flow arrival follows poison distribution, which may not be truly necessary for practice [88].

For more intelligent flow deletion strategy, the work of Yang et al. [88] proposed smart table entry eviction for OpenFlow switches (STEREOS). The ML-based approach was used to classify flow entries as either active or inactive. On this basis, intelligence is used to decide the right flow to be evicted without having to pay for the storage overhead. Li et al. [96] noted that due to high storage load, the flowtable is exploited, which in turn affects the performance of the data plane. Toward this goal, they proposed HQTimer, a Q-learning base for the selection of flow effective timeout values to improve the performance of the data plane. However, these approaches can offer better performance in a small–medium scale network, but a dynamic large-scale network may require a more sophisticated training set, which in turn needs more storage to accommodate more historical data. While switch storage is scarce, resources and controller CPU is also limited. Considering the

built-in features in the OpenFlow network and the global networking knowledge of the controller. It is important to leverage the features and achieve better routing optimization, QoS provision, traffic flow classification, and network resource management. Therefore, more research work is required with features selections to predict flows and decide on the right flow to preserve in the switch flowtable in case of a large-scale network.

## 5. Challenges and Future Research Direction

The separation of the control plane from the data plane offered many network solutions; however, it poses many network challenges, among which include overhead, security threat, delay. Similarly, the limited switch memory also introduces several challenges such as security threat, higher update operation, which can affect the performance of SDN, especially in large-scale networks. Different efforts were made in recent years with different solutions for various use cases. Few research works have discussed the performance of SDN in dynamic and large scale-networks like Telco, where dense traffic flows are generated regularly. Such an environment is shown to have a massive number of flows containing a large number of packets at a short time interval. However, switch memory is constrained and cannot accommodate the required number of flow entries. Managing massive flows with a large number of packets in the limited switch flowtable remains as one of the challenging researches, which deserve more research efforts.

### 5.1. Reactive and Proactive Flow Table Rule Installation

Previous sections have discussed the potentialities of both reactive and proactive flowtable rule allocation approaches. Table 6 summarizes the challenges associated with each approach. Therefore, efficient flow rules allocation should combine both reactive and proactive to improve the flowtable usage and reduce overhead, packet delay. It is recommended that re-routing rules [7] must be completed within 25 milliseconds (ms) to meet stringent QoS requirements of real-time applications. One way is to use proactive entries allocation for delay-sensitive application. While non-delay application or best-effort traffic should benefit from the reactive approach. It will be an interesting area of research to devise a scheme considering traffic variabilities.

**Table 6.** Reactive and proactive rule allocation challenges.

| Issues | Reactive | Proactive |
|---|---|---|
| Switch flowtable TCAM resource | More storage | Space limitation |
| Frequent and dynamic flowtable network | Frequent | Less |
| Packet processing delay | High | Low |
| Packet losses | Proportional to the usage of flowtable: low | Proportional to the usage of flowtable. High may increase the chance of overflow, which leads to more packet losses |
| TCAM update operation | Less because flows are installed on demand | Hard with a significant delay because flow is installed in advance |
| Controller overhead | Higher overhead because of the frequent controller consultation | Less because already installed |
| Switch overhead | Low | High |
| Scalability | Scalable for a large network | Scalable for a small network |

*5.2. Intelligent Flow Table Management*

In large scale networks, a large number of traffic flows are generated more often, which in turn require corresponding flow rules to be installed, and thus, consumed large storage space. Most of the existing solutions to address the storage concern have been tested on small scale networks, for example between (7–28 nodes and 7–43 links) [15]. Thus, these numbers are incomparable to a large number of devices in a large-scale network environment like Telco, Internet of a Thing (IoT). Machine learning (ML) techniques are quite effective in resource management (i.e., network components like switches and controller(s)). However, most of the ML techniques focus on flow classification, flow monitoring. Little research work focuses on predicting traffic flow for real-time application and best-effort traffic and decides on which traffic flow to be installed in advance. OpenFlow has built-in data collection, which stored information about flows such as packet count, byte count. This statistical information indicates the frequency of traffic flows. Therefore, it will be interesting to devise a scheme that may leverage on this build in data collection to devise a scheme that will further reduce the flowtable memory space usage and speed up flow matching rate possibly using fuzzy theory in the selection of frequently used flow rules to be placed in the flowtable.

*5.3. Flow Rule Update Operation*

TCAM support exact and wildcard matching rule, in contrast to exact, wildcard rule makes multiple flows aggregately, thereby reusing entries among different flows, hence, minimized number of entries and reduce overhead for frequent flow setup request. However, TCAM is slow in flow entries update operation. As a result, packets incurred significant delays, especially in large-scale networks. For example, failure recovery has a strict recovery delay requirement to adhere to the carrier-grade network. As such, minimum number of rule updates are preferable in network with a large number of devices. Single switch rules update has received some attention; however, multi-switch flowtable update received little attention. A further analytical model is required to investigate and reduce the packet processing delay mainly caused by the update operation. It will be an interesting research to leverage on multiple flow table pipelining features to flexibly devise a multi-switch flowtable rule update optimization scheme to further reduce the update delay.

## 6. Conclusions

Software defined networking is an emerging paradigm that offers network flexibility and management of flows. This flexibility comes at the cost of smaller flowtable capacity, which introduces switch and controller overhead concern. To this end, it is important to improve the efficiency of the limited flow table, thereby reducing the controller and switch overhead. In this way, this survey presents the state-of-the-art pieces of work with different solutions to improve the limited flowtable, which in turn introduces SDN performance concern. Other possible ways are to explore the use of fuzzy theory and machine learning techniques in selecting the frequent flow entries to be preserved in the flow table. Therefore, challenges have been identified in this paper, which include update operation, resource limitations, communication overhead, and packet processing delay. Ideas and solutions to decisively address these concerns were discussed, and finally, potential research direction was pointed out. Thus, more research work is required to address the remaining gaps.

**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. Backing, N.T.A. *Annual Report 2019*; Investing in Australia; 2019. Available online: http://www.pinnacleinvestment.com (accessed on 20 February 2020).
2. Kreutz, D.; Ramos, F.M.V.; Verissimo, P.; Rothenberg, C.E.; Azodolmolky, S.; Member, S.; Uhlig, S. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* **2014**, *103*, 14–76. [CrossRef]
3. 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]
4. Open Networking Foundation. *SDN Architecture Overview*; ONF: Palo Alto, CA, USA, 2013; Volume 1, pp. 1–5. Available online: www.opennetworking.org (accessed on 5 January 2020).
5. Al-fares, M.; Radhakrishnan, S.; Raghavan, B.; Huang, N.; Vahdat, A. Hedera: Dynamic Flow Scheduling for Data Center Networks. *Nsdi* **2010**, *10*, 89–92.
6. Jain, S.; Kumar, A.; Mandal, S.; Ong, J.; Poutievski, L.; Singh, A.; Venkata, S.; Wanderer, J.; Zhou, J.; Zhu, M.; et al. B4: Experience with a Globally-Deployed Software Defined WAN. *ACM SIGCOMM Comput. Commun. Rev.* **2013**, *43*, 3–14. [CrossRef]
7. Qiu, K.; Member, S.; Yuan, J.; Zhao, J.; Wang, X.; Secci, S.; Member, S.; Fu, X.; Member, S. FastRule: Efficient Flow Entry Updates for TCAM-Based OpenFlow Switches. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 484–498. [CrossRef]
8. 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]
9. Curtis, A.R.; Mogul, J.C.; Tourrilhes, J.; Yalagandula, P.; Sharma, P.; Banerjee, S. DevoFlow: Scaling Flow Management for High-Performance Networks. In Proceedings of the ACM SIGCOMM 2011 Conference, Toronto, ON, Canada, 15–19 August 2011.
10. Wang, D.; Li, Q.; Jiang, Y.; Xu, M.; Hu, G. Balancer: A Traffic-Aware Hybrid Rule Allocation Scheme in Software Defined Networks. In Proceedings of the 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, Canada, 31 July–3 August 2017.
11. Katta, N.; Alipourfard, O.; Rexford, J.; Walker, D. CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks Categories and Subject Descriptors. In Proceedings of the Symposium on SDN Research, Santa Clara, CA, USA, 14–15 March 2016.
12. Yu, M.; Rexford, J.; Freedman, M.J.; Wang, J. Scalable Flow-Based Networking with DIFANE. *ACM SIGCOMM Comput. Commun. Rev.* **2010**, *40*, 351–362. [CrossRef]
13. Kotani, D.; Okabe, Y. A Packet-In Message Filtering Mechanism for Protection of Control Plane in OpenFlow Switches. In Proceedings of the 2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), Marina del Rey, CA, USA, 20–21 October 2014; pp. 695–707.
14. Dridi, L.; Zhani, M.F. SDN-Guard: DoS Attacks Mitigation in SDN Networks. In Proceedings of the 2016 5th IEEE International Conference on Cloud Networking (Cloudnet), Pisa, Italy, 3–5 October 2016.
15. Kandoi, R.; Antikainen, M. Denial-of-Service Attacks in OpenFlow SDN Networks. In Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, Canada, 11–15 May 2015.
16. Dridi, L.; Zhani, M.F. A holistic approach to mitigating DoS attacks in SDN networks. *Int. J. Netw. Manag.* **2018**, *28*, 1–14. [CrossRef]
17. Vissicchio, S.; Cittadini, L. Safe, Efficient, and Robust SDN Updates by Combining Rule Replacements and Additions. *IEEE/ACM Trans. Netw.* **2017**, *25*, 3102–3115. [CrossRef]
18. Zhao, B.; Zhao, J.; Wang, X.; Wolf, T.; Member, S.; Software-defined, A. RuleTailor: Optimizing Flow Table Updates in OpenFlow Switches With Rule Transformations. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 1581–1594. [CrossRef]
19. Luo, H.; Li, W. Mitigating SDN Flow Table Overflow. In Proceedings of the 2018 IEEE 42nd Annual Computer Software Application Confeence, Tokyo, Japan, 23–27 July 2018; Volume 1, pp. 821–822.

20. Kim, E. Enhanced Flow Table Management Scheme With an LRU-Based Caching Algorithm for SDN. *IEEE Access* **2017**, *5*, 25555–25564. [CrossRef]

21. Kim, N.; Kim, D.; Jang, Y.; Lee, C.; Lee, B. Applied sciences Traffic Characteristics in Software-Defined Networks. *Appl. Sci.* **2020**, *10*, 3590. [CrossRef]

22. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69. [CrossRef]

23. Martinez, C.; Ferro, R.; Ruiz, W. Next generation networks under the SDN and OpenFlow protocol architecture. In Proceedings of the 2015 Workshop on Engineering Applications-International Congress on Engineering (WEA), Bogota, Colombia, 28–30 October 2015.

24. Bianchi, G.; Bonola, M.; Capone, A.; Cascone, C. OpenState: Programming Platform-independent Stateful OpenFlow Applications Inside the Switch. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 44–51. [CrossRef]

25. Malik, A.; Aziz, B.; Al-haj, A.; Adda, M. Software-Defined Networks: A Walkthrough Fault Tolerance. *Peer J. Prepr.* **2019**. [CrossRef]

26. *The Benefits of Multiple Flow Tables and TTPs*; Version Number 1.0, Tr-510, O.N.F.; 2015; pp. 1–9. Available online: https://www.opennetworking.org/wp-content/uploads/2014/10/TR_Multiple_Flow_Tables_and_TTPs.pdf (accessed on 2 March 2020).

27. Narisetty, R.; Dane, L.; Malishevskiy, A.; Gurkan, D.; Bailey, S.; Narayan, S.; Mysore, S. OpenFlow configuration protocol: Implementation for the of management plane. In Proceedings of the 2013 Second GENI Research and Educational Experiment Workshop, Salt Lake City, UT, USA, 20–22 March 2013; pp. 66–67.

28. Liu, B. Exposing End-to-End Delay in Software-Defined Networking. *Int. J. Reconfig. Comput.* **2019**, *2019*, 7363901.

29. Sharma, S.; Staessens, D.; Colle, D.; Pickavet, M.; Demeester, P. Fast failure recovery for in-band OpenFlow networks. In Proceedings of the 2013 9th International Conference on the Design of Reliable Communication Networks (DRCN), Budapest, Hungary, 4–7 March 2013; pp. 52–59.

30. Fernandez, M.P. Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive. In Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), Barcelona, Spain, 25–28 March 2013.

31. Li, R.; Wang, X. A Tale of Two ( Flow ) Tables: Demystifying Rule Caching in OpenFlow Switches. In Proceedings of the 48th International Conference on Parallel Processing, Kyoto, Japan, 5–8 August 2019.

32. Favaro, A.; Ribeiro, E.P. Reducing SDN/OpenFlow Control Plane Overhead with Blackhole Mechanism. In Proceedings of the 2015 Global Information Infrastructure and Networking Symposium (GIIS), Guadalajara, Mexico, 28–30 October 2015.

33. Vishnoi, A.; Poddar, R.; Mann, V.; Bhattacharya, S. Effective Switch Memory Management in OpenFlow Networks. In Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, DEBS '14, Mumbai, India, 26–29 May 2014.

34. 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.

35. Jin, X.; Liu, H.H.; Rexford, J.; Wattenhofer, R. Dynamic Scheduling of Network Updates. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 539–550. [CrossRef]

36. Ding, Z.; Fan, X.; Yu, J.; Bi, J. Update Cost-Aware Cache Replacement for Wildcard Rules in Software-Defined Networking. In Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 25–28 June 2018; pp. 457–463.

37. Qiu, K.; Yuan, J.; Zhao, J.; Wang, X.; Secci, S.; Fu, X. Fast Lookup Is Not Enough: Towards Efficient and Scalable Flow Entry Updates for TCAM-based OpenFlow Switches. In Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, 2–6 July 2018; pp. 918–928.

38. Wen, X.; Yang, B.; Chen, Y.; Li, L.E.; Bu, K.; Zheng, P.; Yang, Y.; Hu, C. RuleTris: Minimizing Rule Update Latency for TCAM-based SDN Switches. In Proceedings of the 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), Nara, Japan, 27–30 June 2016; pp. 179–188.

39.  Huang, X. Tango: Simplifying SDN Control with Automatic Switch Property Inference, Abstraction, and Optimization Categories and Subject Descriptors. In Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, CoNEXT '14, Sydney, Australia, 2–5 December 2014.

40.  Sabih, A.F. Cognitive Smart Agents for Optimising OpenFlow Rules in Software Defined Networks. Ph.D. Dissertation, Brunel University, London, UK, 2017.

41.  Yao, J.; Han, Z.; Sohail, M. A Robust Security Architecture for SDN-Based 5G Networks. *Future Internet* **2019**, *11*, 85. [CrossRef]

42.  Polat, H.; Polat, O. Detecting DDoS Attacks in Software-Defined Networks Through Feature Selection Methods and Machine Learning Models. *Sustainability* **2020**, *12*, 1035. [CrossRef]

43.  Ye, J.; Cheng, X. A DDoS Attack Detection Method Based on SVM in Software Defined Network. *Secur. Commun. Netw.* **2018**, *2018*, 9804061. [CrossRef]

44.  Li, X.; Yuan, D.; Hu, H.; Ran, J.; Li, S. DDoS Detection in SDN Switches using Support Vector Machine Classifier. In Proceedings of the 2015 Joint International Mechanical, Electronic and Information Technology Conference, Chongqing, China, 18–20 December 2015; pp. 344–348.

45.  Nanda, S.; Zafari, F.; Decusatis, C.; Wedaa, E.; Yang, B. Predicting Network Attack Patterns in SDN using Machine Learning Approach. In Proceedings of the 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Palo Alto, CA, USA, 7–10 November 2016.

46.  Zhang, L.; Wang, S.; Xu, S.; Lin, R.; Yu, H. TimeoutX: An Adaptive Flow Table Management Method in Software Defined Networks. In Proceedings of the 2015 IEEE Global Communications Conference (GLOBECOM), San Diego, CA, USA, 6–10 December 2015; pp. 1–6.

47.  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.

48.  Challa, R.; Lee, Y.; Choo, H. Intelligent Eviction Strategy for Efficient Flow Table Management in OpenFlow Switches. In Proceedings of the 2016 IEEE NetSoft Conference and Workshops (NetSoft), Seoul, Korea, 6–10 June 2016; pp. 312–318.

49.  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.

50.  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. In Proceedings of the IOP Conference Series: Materials Science and Engineering, Zhangjiajie, China, 10–12 August 2019.

51.  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.

52.  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]

53.  Li, X.; Huang, Y. A Flow Table with Two-Stage Timeout Mechanism for SDN Switches. 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. 1804–1809.

54.  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.

55.  Qian, S.; 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.

56.  Rifai, M.; Huin, N.; Caillouet, C.; Giroire, F.; Moulierac, J.; Lopez, D.; Urvoy-keller, G. Minnie: An SDN world with few compressed forwarding rules. *Comput. Netw.* **2017**, *121*, 185–207. [CrossRef]

57. 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.

58. Zarek, A.; Ganjali, Y.; Lie, D. *OpenFlow Timeouts Demystified*; University of Toronto: Toronto, ON, Canada, 2012.

59. 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]

60. 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.

61. Leng, B.; Huang, L.; Qiao, C.; Xu, H.; Wang, X. FTRS: A mechanism for reducing flow table entries in software defined networks R. *Comput. Netw.* **2017**, *122*, 1–15. [CrossRef]

62. Meiners, C.R.; Liu, A.X.; Torng, E. Bit Weaving: A Non-Pre fi x Approach to Compressing Packet Classi fi ers in TCAMs. *IEEE/ACM Trans. Netw.* **2012**, *20*, 488–500. [CrossRef]

63. Li, X.; Shao, Y. Memory compression for Recursive Flow Classification Algorithm in Network Packet Processing Devices. In Proceedings of the 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chongqing, China, 12–14 October 2018; pp. 1502–1505.

64. Tsai, T. Dynamic Flow Aggregation in SDNs for Application-aware Routing. In Proceedings of the International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP), Prague, Czech Republic, 20–22 July 2016; pp. 1–5.

65. Luo, S.; Yu, H.; Li, L.M. Fast Incremental Flow Table Aggregation in SDN. In Proceedings of the 2014 23rd International Conference on Computer Communication and Networks (ICCCN), Shanghai, China, 4–7 August 2014; pp. 1–8.

66. Chao, T.; Wang, K. In-switch Dynamic Flow Aggregation in Software Defined Networks. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–6.

67. Wang, C.; Youn, H.Y. Entry Aggregation and Early Match Using Hidden Markov Model of Flow Table in SDN. *Sensors* **2019**, *19*, 2341. [CrossRef]

68. Kannan, K.; Banerjee, S. Compact TCAM: Flow Entry Compaction in TCAM for Power Aware SDN. In *International Conference on Distributed Computing and Networking*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 439–440.

69. Kang, N.; Liu, Z.; Rexford, J.; Walker, D. Optimizing the "One Big Switch" Abstraction in Software-Defined Networks. In Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, Santa Barbara, CA, USA, 13 December 2013.

70. Kanizo, Y.; Hay, D.; Keslassy, I.; Background, A. Palette: Distributing Tables in Software-Defined Networks. In Proceedings of the 2013 IEEE INFOCOM, Turin, Italy, 14–19 April 2013; pp. 545–549.

71. Nguyen, X.; Saucez, D.; Barakat, C.; Turletti, T. OFFICER: A general Optimization Framework for OpenFlow Rule Allocation OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement. In Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM), Kowloon, Hong Kong, 26 April–1 May 2015.

72. Sheu, J.; Lin, W.; Chang, G. Efficient TCAM Rules Distribution Algorithms in Software-Defined Networking. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 854–865. [CrossRef]

73. Sheu, J.; Wang, P.; Rb, J. Wildcard-Rule Caching and Cache Replacement Algorithms in Software-Defined Networking. In Proceedings of the 2017 European Conference Networks Communications, Oulu, Finland, 12–15 June 2017; pp. 1–6.

74. Huang, J.; Chang, G.; Wang, C.; Lin, C. Heterogeneous Flow Table Distribution in Software-Defined Networks. *IEEE Trans. Emerg. Top. Comput.* **2016**, *4*, 252–261. [CrossRef]

75. 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]

76. Mohan, P.M.; Truong-huu, T.; Gurusamy, M. Fault tolerance in TCAM-limited software defined networks. *Comput. Netw.* **2017**, *116*, 47–62. [CrossRef]

77. Li, X.; Xie, W. CRAFT: A Cache Reduction Architecture for Flow Tables in Software-Defined Networks. In Proceedings of the 2017 IEEE Symposium on Computers and Communications (ISCC), Heraklion, Greece, 3–6 July 2017.

78. Yan, B.; Xu, Y.; Xing, H.; Xi, K.; Chao, H.J. CAB: A Reactive Wildcard Rule Caching System for Software-Defined Networks Reactively Caching Rules on Demand. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, Chicago, IL, USA, 22 August 2014.

79. Katta, N.; Alipourfard, O.; Rexford, J.; Walker, D. Infinite CacheFlow in Software-Defined Networks. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, Chicago, IL, USA, 22 August 2014.

80. Wu, J.; Chen, Y.; Zheng, H. Approximation Algorithms for Dependency-Aware Rule-Caching in Software-Defined Networks. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, UAE, 9–13 December 2018; pp. 1–6.

81. Wang, L.; Li, Q.; Sinnott, R.; Jiang, Y.; Wu, J. An intelligent rule management scheme for Software Defined Networking. *Comput. Netw.* **2018**, *144*, 77–88. [CrossRef]

82. Cheng, T.; Wang, K.; Wang, L.C.; Lee, C.W. An In-switch Rule Caching and Replacement Algorithm in Software Defined Networks. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6.

83. Al-fuqaha, A.L.A.; Shuaib, K.; Sallabi, F.M.; Qadir, J. SDN Flow Entry Management Using Reinforcement. *ACM Trans. Auton. Adapt. Syst. TAAS* **2018**, *13*, 1–23.

84. OpenFlow Reference Switch Specification. *Current* **2009**, 1–36. Available online: http://www.openflow.org/ (accessed on 20 February 2020).

85. Amaral, P.; Bernardo, L. Machine Learning in Software Defined Networks: Data Collection and Traffic Classification. In Proceedings of the 2016 IEEE 24th International Conference on Network Protocols (ICNP), Singapore, 8–11 November 2016; pp. 1–5.

86. Sminesh, C.N.; Kanaga, E.G.M.; Ranjitha, K. Flow Monitoring Scheme for Reducing Congestion and Packet Loss in Software Defined Networks. In Proceedings of the 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 6–7 January 2017; pp. 1–5.

87. Yu, C. DROM: Optimizing the Routing in Software-Defined Networks With Deep Reinforcement Learning. *IEEE Access* **2018**, *6*, 64533–64539. [CrossRef]

88. 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]

89. Rossi, D.; Valenti, S. Fine-grained traffic classification with Netflow data. In Proceedings of the 6th International Wireless Communications and Mobile Computing Conference, IWCMC '10, Caen, France, 28 June–2 July 2010; pp. 479–483.

90. Xie, J.; Yu, F.R.; Huang, T.; Xie, R.; Liu, J.; Wang, C.; Liu, Y. A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 393–430. [CrossRef]

91. Kumar, S.; Louis, S.; Louis, S.; Crowley, P.; Turner, J. Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection. *ACM SIGCOMM Comput. Commun. Rev.* **2006**, *36*, 339–350. [CrossRef]

92. Glick, M.; Rastegarfar, H. Scheduling and Control in Hybrid Data Centers. In Proceedings of the 2017 IEEE Photonics Society Summer Topical Meeting Series (SUM), San Juan, Puerto Rico, 10–12 July 2017.

93. Xiao, P.; Qu, W.; Qi, H.; Xu, Y.; Li, Z. An Efficient Elephant Flow Detection with Cost-Sensitive in SDN. In Proceedings of the 2015 1st International Conference on Industrial Networks and Intelligent Systems (INISCom), Tokyo, Japan, 2–4 March 2015; pp. 24–28.

94. 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.

95. Kannan, K.; Banerjee, S. FlowMaster: Early Eviction of Dead Flow on SDN. In Proceedings of the International Conference on Distributed Computing and Networking, Coimbatore, India, 5–8 January 2014; pp. 484–498.

96. 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]