IEEE Access
Multidisciplinary | Rapid Reviews | Open Access Journal

Received July 1, 2019, accepted July 23, 2019, date of publication August 1, 2019, date of current version August 19, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2932422

Toward Adaptive and Scalable OpenFlow-SDN Flow Control: A Survey

MOHAMMED ALSAEEDI[1], MOHD MURTADHA MOHAMAD[1], AND ANAS A. AL-ROUBAIEY[2]

[1]School of Computing, Faculty of Engineering, Universiti Teknologi Malaysia, Johor Bahru 81310, Malaysia
[2]Computer Engineering Department, College of Computer Science and Engineering, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia

Corresponding author: Mohammed Alsaeedi (amamohammed22@live.utm.my)

ABSTRACT Software-defined networking (SDN) is an emerging network architecture that promises to simplify network management, improve network resource utilization, and boost evolution and innovation in traditional networks. The SDN allows the abstraction and centralized management of the lower-level network functionalities by decoupling the network logic from the data forwarding devices into the logically centralized distributed controllers. However, this separation introduces new scalability and performance challenges in large-scale networks of dynamic traffic and topology conditions. Many research studies have represented that centralization and maintaining the global network visibility over the distributed SDN controller introduce scalability concern. This paper surveys the state-of-the-art proposed techniques toward minimizing the control to data planes communication overhead and controllers' consistency traffic to enhance the OpenFlow-SDN scalability in the context of logically centralized distributed SDN control plane architecture. The survey mainly focuses on four issues, including logically centralized visibility, link-state discovery, flow rules placement, and controllers' load balancing. In addition, this paper discusses each issue and presents an updated and detailed study of existing solutions and limitations in enhancing the OpenFlow-SDN scalability and performance. Moreover, it outlines the potential challenges that need to be addressed further in obtaining adaptive and scalable OpenFlow-SDN flow control.

INDEX TERMS SDN, OpenFlow, controller, scalability, global network view, flow rules placement, centralized flow control, load balancing, discovery protocol.

I. INTRODUCTION

Today's Internet is used as a global communication platform for the heterogeneous and large number of dynamic applications, services, physical objects, and machines. Network traffic control and orchestration in modern networks is a very complex task that requires to adapt to the time-varying changes in link utilization, bandwidth allocation, latency, energy consumption, and jitter over a heterogeneous network. The emerging Internet of Things (IoT) and the adoption of multi-tenant data centers (DC) generate a large amount of traffic and add more complexity to the network. Unfortunately, traditional network's architecture is complex and not well designed to enable the fine-grained and QoS-aware traffic engineering over the network. The compact integration of control and data planes complicates the network traffic monitoring process resulted in less QoS-aware flow control and inefficient resource utilization. It is time-consuming and expensive to manage the network devices separately especially in time-varying and multi-tenant data center environments. According to the Enterprise Strategy Group (ESG), traditional networks require to automate the manual processes of network management, provides better visibility for efficient resource utilization and provides dynamic network orchestration to align with cloud computing. ESG stats that as data center grows in scale, network management operators struggle of too many manual and reconfiguration processes, which may reach nearly 40 percent of the most common network operations problems.

In addition to the configuration complexity, traditional networks are not well designed to adapt and self-manage to active, unpredictable faults and load changes in large-scale networks [1]. The underline network architecture lacks programmability, and hence cannot meet the application layer needs in real-time. The time-varying and tremendous amount of traffic in application layer requires global network

The associate editor coordinating the review of this manuscript and approving it for publication was Yu-Chi Chen.

visibility and abstraction for better QoS provisioning, resource utilization and for avoiding the need to enforce global policies by carefully crafting switch-by-switch configuration.

Software-Defined Networking (SDN) is an emerging network architecture that promises to simplify network management, improve network resource utilization, and boost evolution and innovation in traditional networks. SDN introduces the abstraction on the network layers by separating the control plane (networking logic) from the data plane (routers and switches) to an external entity (*controller*). This separation decouples network devices from management and allows both planes to evolve independently, provides design flexibility and programmability compared to traditional network architectures [2].

In SDN architecture, traffic forwarding devices become simpler and easier to be deployed, upgraded or configured by a centralized SDN controller. The network management and configuration become much simpler as they can be performed from a centralized point in the network and thus significantly reduce the operational expenses (OPEX). The centralized management of SDN enables dynamic on the fly reconfiguration and adaptation to the time-variant fault and load changes in the network as well as reduces the complexity of routing and traffic engineering. SDN simplifies orchestration for both static and dynamic network changes. It also enables a standard based homogeneous network and network programmability, portability, global management and optimization, and efficient utilization of the network resources. Furthermore, it provides true network visualization by enabling the abstraction of the underline network and offers a flexible Network as a Service (NaaS) as proposed in OpenVirtex [3]. Virtual networks can also efficiently slice and share the same physical hardware resources to be provisioned on-demand without affecting virtual network operations [4].

Today SDN gains more interest in the industry. According to Statistics MRC, ''the Global Software Defined Networking (SDN) Market accounted for $10.88 billion in 2015 and is expected to reach $134.51 billion by 2022 growing at a CAGR of 43.2% from 2015 to 2022'', which is extremely high. Google, as an instance, uses SDN in its data centers' network solutions such as B4 [5], Jupiter [6] and Andromeda. Google's B4 is a network backbone that has adopted SDN for interconnecting WAN data centers across the planet. The reason behind using SDN in B4 is to increase efficient link utilization and enable global network visibility over the network edge. Thus, it promotes relative application traffic demands/priorities during resource constraints, dynamically reassign bandwidth in response to link failures or changes in traffic patterns and control bursts traffic rather than a complex over-provisioning. B4 shows an increase in link utilization reaches to an average of 70% utilization which corresponding to $2-3x$ efficiency of standard network practice.

Despite the advantages of decoupling the network control logic from the underlying forwarding devices, there have been concerns on the network scalability and performance. The new communication channel between control and data planes adds extra delay and increases the amount of control traffic, which can be vertical between the control and data planes or horizontal among the distributed controllers. This amount of traffic increases proportionally to the time-varying changes and scaling of the network, resulting in high communication and computational load in the control plane [7]–[9].

Several studies have been conducted toward improving SDN-based networks scalability and performance. Early works focused on restructuring the control plane by distributing the controllers hierarchically or horizontally while maintaining a logically centralized control on each distributed controller [10]–[16]. Other research works focus on the elasticity and placement of the distributed controllers to dynamically identify the optimal number and locations of controllers in the network [17]–[22]. Researchers have also considered the deployment of SDN switches and their controllers in hybrid SDNs for maximizing the number of flows managed by SDN and therefore enhancing legacy network's scalability and performance [23]–[32]. However, in this survey we focus on the research studies toward re-engineering SDN communication traffic vertically between control and data planes or horizontally among controllers, resulted from some SDN-scalability related challenges such as 1) controller's global visibility, 2) link-state discovery, 3) flow-rules placement, and 4) controllers' load unbalancing, in dynamic and large-scale networks. Considering the importance of SDN flow management in the future of wire/wireless networks, this paper presents a comprehensive literature survey on some of the key challenges and research efforts to enhance the OpenFlow-SDN scalability and performance in the context of logically-centralized distributed SDN controllers. It also provides a discussion and comparison study on the currently proposed solutions and demonstrates some future research challenges.

The rest of the paper organizes as follows: Section II gives an overview of the standard OpenFlow-SDN flow control. Section III presents the key challenges in the OpenFlow-SDN flow control that affect the performance and scalability of the network. Section IV provides a detailed survey of existing and state-of-the-art solutions proposed in the literature of the OpenFlow-SDN logically centralized visibility. Section V provides a detailed survey of existing and state-of-the-art solutions proposed in the literature of the OpenFlow-SDN link-state discovery. Section VI provides a detailed survey of existing and state-of-the-art solutions proposed in the literature of the OpenFlow-SDN flow-rules placement. Section VII provides a detailed survey of existing and state-of-the-art solutions proposed in the literature of the OpenFlow-SDN controllers load balancing. Section VIII discusses and summarizes the research challenges and future directions. Finally, Section IX wraps the paper up with concluding remarks.

II. AN OVERVIEW OF OPENFLOW-SDN FLOW CONTROL

SDN architecture consists of five main components; application plane (AP), northbound interface (NBI),

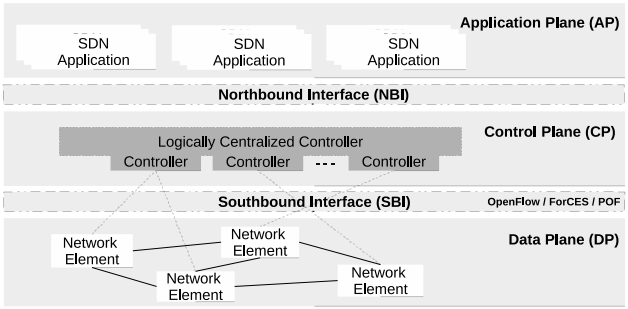

FIGURE 1. Physically distributed but logically centralized SDN architecture.

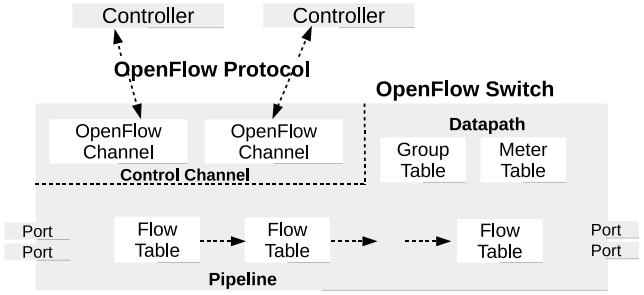

FIGURE 2. OpenFlow version 1.5 switch components [34].

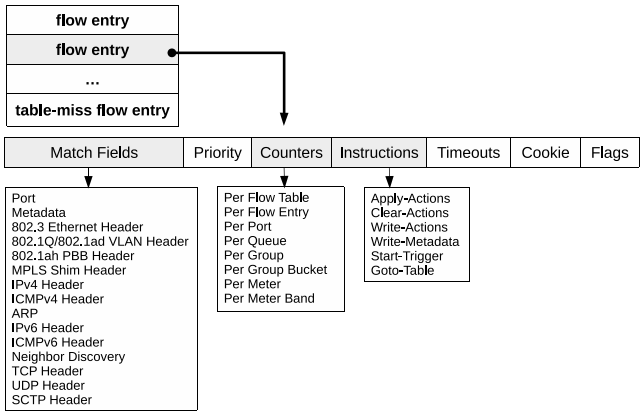

FIGURE 3. OpenFlow 1.5 flow entry [34].

control plane (CP), southbound interface (SBI) and data plane (DP) [33] as shown in Figure 1. The application plane is the set of network applications that leverage the exposed northbound APIs to define the rules and instructions that control the network logic. The northbound APIs interface translates these instructions to the control plane which provides fine-grained control over the forwarding nodes and offers many network services such as routing, monitoring, load balancers, and firewalls. These applications either embodied in the control plane (e.g., routing optimization, network management and monitoring, security, traffic engineering, and QoS control) or located in a proxy server (e.g., firewall and authentication servers). The control plane consists of one or more controllers that forward the instruction sets and policies defined by network applications to the data plane via the southbound APIs interface. Centralization and visibility over the network allow the controller to orchestrate application demands for limited dynamic network resources [33].

OpenFlow [35] is the first and predominant SDN flow control protocol which has already been the de facto standard for controlling SDN-based switches. It plays the role of the southbound interface to allow the controller to have direct access and control of the data forwarding network devices. OpenFlow is standardized by Open Networking Foundation (ONF) to address dynamic nature and high-bandwidth of today's applications, and reduce the management complexity. Forwarding and Control Elements (ForCES) [36] and Protocol-oblivious forwarding (POF) are other examples of the southbound flow control platforms. Similarly to the flow tables in OpenFlow, ForCES uses logical function blocks (LFB) in the data forwarding devices to provide networking functionalities, such as IP routing. However, this research aims to study the state-of-the-art flow control mechanisms and research challenges in the OpenFlow-SDN. The lowest layer of SDN architecture is the data plane that consists of simple physical/virtual data forwarding nodes such as switches or vSwitch. This layer includes the minimal necessary network functions as packets lookups and forwarding in which they are responsible for forwarding packets according to the rules and instructions that are configured by the control plane.

According to the OpenFlow Switch Specification version 1.5, an OpenFlow-based virtual or physical switch consists of at least one ingress flow table, a group table and control channels to communicate with the controller as shown in Figure 2. Flow tables are sequentially ordered and can perform packet lookups and forwarding. Each flow table can store a set of flow entries that consists of matching fields, statistical counters and a set of flow instructions (actions) as illustrated in Figure 3. Matching fields are used to match the packet header fields such as Ethernet source and destination address, packet ingress port, and other pipeline fields. The matching field can use wild-card to match any value or in some cases uses bit-masked to match a subset of bits. The controller can send/receive events from/to the data forwarding node via the OpenFlow-based control channels. It can also add, update or remove the installed flow entries reactively or proactively. If a packet arrives on an OpenFlow-based forwarding node, the matching process starts in the first flow table and may continue on the next tables in the pipeline unless a matching flow entry is found. The process of checking flow entries in the pipeline flow tables is performed sequentially. When the packet's matching header matches with a flow entry in the pipeline flow tables, the instructions associated with the flow entry will be executed. Otherwise, the instruction associated with the table-miss flow entry in the last flow table will be executed and depending on the configuration of the table-miss flow entry, the packet either be forwarded to the controller over the OpenFlow channel or simply dropped.

The instructions associated with each flow entry either contain actions or pipeline processing. Actions are responsible for giving instructions to packet modification, forwarding

or group table processing. On the other hand, pipeline processing is responsible for giving instructions to sent packets to the subsequent flow tables and allows the communication of meta-data information among flow tables. The matched packet may be forwarded to physical, logical or a reserved port defined by the specification. The reserved ports are only used for generic forwarding actions such as broadcasting, sending and receiving to/from the controller, or for non-OpenFlow based forward processing. Moreover, OpenFlow-based data forwarding node contains logical ports that are used to specify link aggregation groups, loopback interfaces or tunnels. Flow packets may also be forwarded to a group table, which specifies additional sets of actions such as flooding or more complex forwarding semantics (e.g., fast reroute, multi-path, and link aggregation). The Group table consists of a set of group entries that contains a list of action buckets with specific semantics dependent on the group type. Furthermore, it enables to forward multiple flow entries to a certain common IP address.

OpenFlow pipeline processing as illustrated in Figure 4 starts when a flow packet arrived at the ingress first flow table and matched against flow entries. If the flow packet does not match with any flow entry, the instruction set direct the packet to the next flow table using the *GotoTable* instruction. The same process will continue checking the matched flow entries in every flow table until the matched flow entry founded and the associated instruction set executed. On the other hand, if pipeline processing completed while no matched flow entry was found, packet then will be processed as a *table_miss*. Depending on the pre-configured instructions in the table-miss entry, the packet either be dropped, forwarded to a subsequent table or sent to the controller on a *packet_in* message via the control channel. In case the table_miss flow entry does not exist, the unmatched packets are dropped (discarded) by default. A large number of packet_in messages are expected to be generated in the case of dynamic and large-scale network traffic conditions where many unknown flow packets frequently arrive at the forwarding node. However, sending a flow request (packet_in message) to the controller for every unknown packet can overwhelm the controller because the controller needs to calculate the forwarding rules of every new packet and then install it to the flow tables in all the respective data forwarding nodes (switches or VSwitches). Such amount of traffic and computational load may lead to a controller overhead and increase flow-rules placement delay, and hence affects the network performance and scalability [37].

In OpenFlow-SDN, the controller has a global network state visibility over the network, which thereby can install the forwarding rules (flow entries) proactively to the flow tables in every connected data forwarding device. However, due to high wildcard lookup performance of Ternary Content-Addressable Memory (TCAMs), it has been used to implement flow tables. Unfortunately, TCAM is very expensive and thus flow tables cannot scale well due to its capacity limitation which usually from 4000 to 32000 entries [1]. According to OpenFlow version 1.5, only a maximum of 1082 bits can be used per flow entry. As a result, flow tables cannot handle a large number of proactively installed flow entries that can provide a network-wide state to the data plane.

III. CHALLENGES AND BACKGROUND

Despite the significant advantages of deploying SDN architecture, the centralization of control plane introduces a major scalability issue for SDN. The single controller architecture of SDN can perform well and obtain optimal flow management and configuration in the case of static and small networks. Controllers like NOX [38], Beacon [39], Floodlight [40], Maestro [41] and McNettle [42] are examples of a centralized single controller. Early research studies focus on improving the performance and scalability of a single controller by exploiting parallelism (e.g., multi-threading, multi-core). For instance, the single-threaded NOX [38] controller is optimized by enabling multi-threading (NOX-MT [43]) to improve its throughput and response time. Beacon [39] and Maestro [41] are also other examples of optimized controllers that use parallelism to improve their performance and scalability. Beacon can achieve a throughput of 12.8 million rps (response per second) with an average delay of 0.02*ms* using 12 processing cores while Maestro can achieve 0.63 million rps with an average delay of 76*ms* using 7 processing cores [37].

Although a single and centralized controller of high computational resources (super controller) can handle a large amount of network flow, it will inevitably form a significant bottleneck in the long run of a large-scale network of dynamic tuning traffic and topology conditions where the number of data plane elements and traffic flow grows over the time. The single centralized SDN controller represents a single point of failure and considerably increases the latency when processing a massive number of data plane requests and controlling the whole network topology. Moreover, in the case of widely separated inter-connected data centers, a single controller introduces a propagation delay. As a result, the idea of multiple controllers is proposed as a solution to resolve the scalability limitation and reliability of a single point of failure controller. In this architecture, the controllers are either logically distributed in one layer (flat) or hierarchically distributed in multi-layer (hierarchical). Table 1 shows some of the popular distributed controllers that can provide centralized management over the network.

Decoupling control logic from data forwarding nodes to allow SDN centralized management and abstraction, introduces new challenging scalability issues. Many research studies have represented that maintaining global network visibility to enable each distributed SDN controller to act as a centralized controller introduces scalability concerns [1], [4], [7]–[9], [49]. The logically centralized visibility can be a major concern for the dynamic and large-scale networks as the Internet of Things (IoT) and Data-Centers (DC) where a huge number of entities (e.g., physical objects,

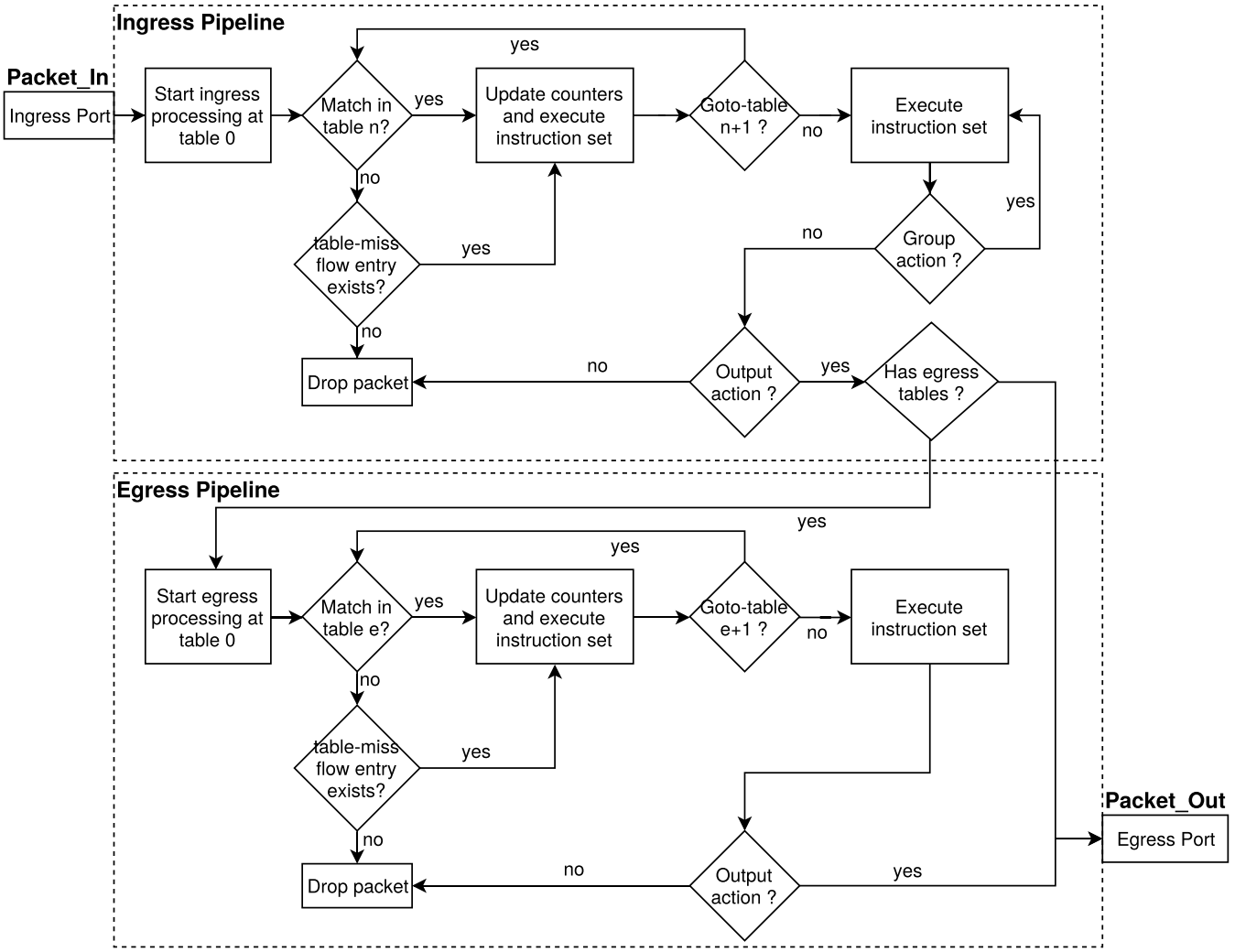

FIGURE 4. OpenFlow 1.5 Pipeline Processing [34].

TABLE 1. OpenFlow-SDN control plane architecture.

	Approach	Description	Pros.	Cons.	Controllers
Single Controller	Physically Centralized	Single controller that manages all the forwarding elements of the network	A bottleneck single point of failure and has scalability limitations	High manageability	NOX [38], Beacon [39], FloodLight [40], Maestro [41] and McNettle [42]
Multiple Controllers	Logically Centralized	Controllers share their local network views to maintain a centralized network-wide state view in every controller	Global view management and higher scalability	Control and consistency overhead	ONIX [10], Beehive [21], HyperFlow [44], ONOS [12], OpenDayLight [13], DISCO [11], SDX [45], RYU [46] and Ravana [47]
	Hierarchically Distributed	Functionalities and view of the controllers are organized based on their level	Distributing traffic load among hierarchical controllers based on network state view requirements (local vs global view)	Traffic filtration may introduce an overhead, high response time	B4 [5], Kandoo [14] and ORION [16], [48]

VMs, applications) communicate and dynamically join or leave the network. As the size of a network grows, controllers become a potential bottleneck and fail to handle all the controller's ingoing/outgoing control or consistency traffic. Therefore, this survey presents some of the scalability challenges facing current OpenFlow-SDN architecture that can lead to control and consistency traffic overhead under dynamic and large-scale network conditions. As illustrated in Figure 5, this survey organizes these challenges into four main categories: Logically Centralized Visibility, Link State Discovery, Flow Rules Placement and Controllers' Load Balancing.

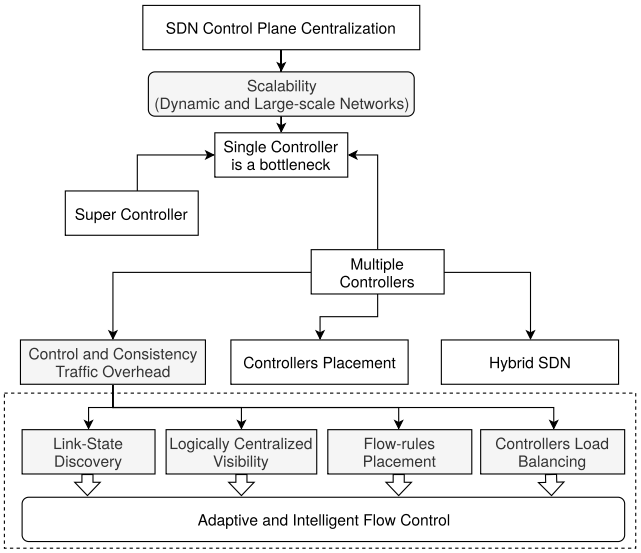

FIGURE 5. OpenFlow-SDN scalability challenges under dynamic and large-scale network conditions.

A. LOGICALLY CENTRALIZED VISIBILITY

In the distributed SDN control plane architecture, every controller requires to have a global view of the network topology to act as a centralized controller. Hence, every controller has to share its local network state view with other controllers. Replicating the global link-state view in every distributed controller can guarantee a logically centralized controlling and transparency over the data plane. However, maintaining a consistent replica of the network-wide state among controllers in a large-scale network of dynamic traffic and topology conditions may result in a massive amount of frequent synchronization traffic that can overwhelm the controller. The interval time between two consecutive synchronizations can also lead to forwarding errors such as routing Loops and black-holes caused by the periodic state inconsistency. Moreover, the time required to maintain a consistent replica of global network state view may result in a spike in the control plane response time and therefore increases the overall flow-rules placement delay and impacts the network scalability and performance [9]. The logically centralized control plane architecture should provide the flexibility to balance the trade-offs between the centralized network-wide state visibility, scalability, and timeliness. The inter-controller consistency process that maintains global network visibility in each controller has to be adaptive to the dynamic network conditions to reduce the amount of consistency traffic that can burden the network scalability.

B. LINK STATE DISCOVERY

In SDN, the control plane has to have a real-time up-to-date view of the global network view and status to act as a centralized controller and efficiently serve the data plane forwarding requests. The entire control plane management procedure is substantially affected by how efficiently it can discover data plane forwarding nodes and links to maintain a centralized global view of the network's topology [50]. The forwarding errors such as Reachability Failures, Routing Loops, Black-holes, Traffic Isolation, and Leakage due to inconsistency between data and control planes proportionally decrease to the efficiency and performance of the discovery process. Furthermore, with the massive proliferation of network devices and the complexity of dynamic and large-scale network topologies, more sophisticated and efficient discovery protocol mechanisms are needed. Therefore, it is important that the SDN discovery protocol supplies the control plane with a real-time view of the network topology to meet the application and dynamic routing QoS demands.

C. FLOW RULES PLACEMENT

Flow-based forwarding nodes use Ternary Content Addressable Memories (TCAMs) to match and forward the arrived packets in constant time. Despite their high speed, TCAMs are expensive and therefore have a minimal capacity. However, SDN traffic routing is per-flow with large flow entries of minimum 356 bits in 15 field tuples out of 40 fields as per the OpenFlow 1.5 specification [34]. To maintain a per-flow fine-grained control, the controller may require to install more than one entry in the flow tables to forward one single flow. Consequently, the OpenFlow-based forwarding element needs a large TCAM memory size to accommodate such a large amount of flow entries.

To cope with the limited size of TCAMs flow tables, controllers can reactively (on-demand) install flow entries every-time a new flow packet arrives at the switch. Unfortunately, in highly dynamic and large-scale networks where traffic changes dynamically over time, the number of flow-rules placement requests increase rapidly, and hence lead to a controller traffic overhead, increase the controller response time and the end-to-end flow-rules placement delay [37]. This delay can significantly increase to a level which can not meet with the requirements of real-time applications and result in degrading network performance and scalability [9].

The scalability and performance drawbacks resulted from the limited size of the flow tables is represented when a high number of new flows aggregated in the edge switches. The data plane needs to forward every first packet of the (unknown) new streams that does not match with any stored flow entry to the controller. As a result, the controller requires to calculate the forwarding rule for each flow (stream) and install it in all corresponding data forwarding devices (switches or routers). Monitor every new stream and install its forwarding rules can overwhelm the controller and makes it as a potential bottleneck [51]. The controller needs to calculate every new flow forwarding path and install it as instructions into the flow table's entries which may add extra latency on routing process and therefore hinder the network performance and scalability, see Figure 6. The reactive placement of forwarding rules can be a serious scalability problem in dynamic large-scale networks such as IoT where a diverse of Internet-connected devices are increased in the

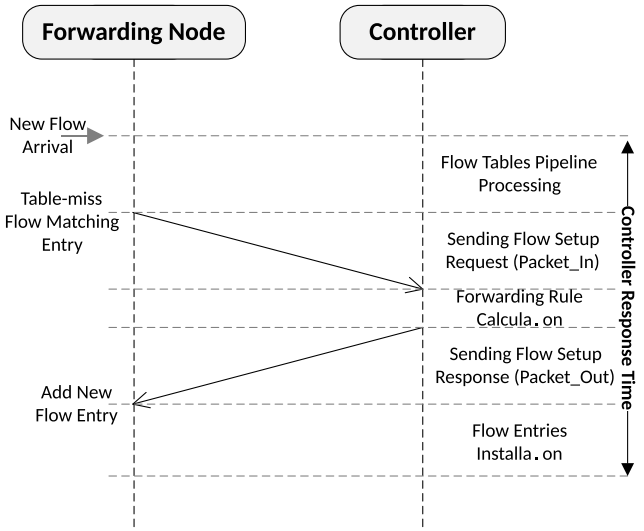

FIGURE 6. OpenFlow-based controller's response time to the flow rules placement request.

volume and variety of network requests, creating a significant load on the controller [52]. IoT networks typically communicate with SDN-based networks through OpenFlow-based gateways and thus add more burden traffic load over the TCAM data tables. Due to the capacity limitation of TCAM, the OpenFlow protocol configures the flow entry's idle timeout with a small value. Therefore, if the flow has not been transmitted within the idle timeout, its entries will be removed to give more space in the flow tables for other active flows. However, the diversity and heterogeneity of IoT communication networks and devices traffic can easily overwhelm the timeout-based flow-rules placement mechanism [53]. The mobility, fluctuation and infrequent transmission of flow samples in IoT based devices make the static idle timeout as an inefficient way to reduce the communication between the controller and OpenFlow-based IoT gateways/sinks because of the removal of IoT flow entries before the arrival of the next IoT flow samples. Results in an additional burden on the controller (controller bottleneck) and thus add extra latency into the overall flow-rules placement process. Furthermore, in mobile or wireless networks where nodes dynamically join and leave the network, a big number of requests are going to be forwarded to the controller to update flow entries which introduce overhead on the control plane and affects the network scalability and performance [37]. The research work in [54] presents the impact of the timeout period on the signaling traffic and the flow table occupancy by studying the inter-arrival times of heterogeneous flows.

D. CONTROLLERS' LOAD BALANCING

In the distributed SDN control plane architecture, the mapping between the data plane elements (switches or routers) and the controllers are statically configured. As a result, controllers can become overloaded due to uneven load balancing of the distributed controllers especially in dynamic large-scale networks of temporal/spatial variations in the traffic characteristics. A controller may become overloaded when the direct connected switches/IoT gateways observe a large amount of aggregated traffic, and hence increase flow-rules placement latency and hinder the network availability and scalability [55]–[57]. Elastic controller provisioning in response to the temporal/spatial variation in network traffic conditions can be used to improve scalability and prevent the controller from being a potential bottleneck [58]. However, inefficient use of the available resources is wasting resources and improperly increases capital expenditures (CAPEX). As a result, the load-balancing among controllers has to be adaptive to the network dynamic changes to maintain better network resource utilization and scalability.

IV. OPENFLOW-SDN LOGICALLY CENTRALIZED VISIBILITY

The distributed controllers require to share their network state view to provide an optimal end-to-end and fine-grained network control and build a consistent and centralized global view of the network. Replicating identical clones of the global network state view in every controller is used to improve reliability, fault-tolerance and (replication) transparency of the distributed controllers. However, periodic synchronization to maintain a consistent replica of the network-wide state among controllers can intensively consume network bandwidth and lead to problems such as controllers overloading or routing misbehavior during the interval between two consecutive synchronizations. Therefore, it is important to synchronize any data plane related event in a timely fashion to keep a consistent network state among the controllers and avoid potential routing misbehaviors such as routing loops and black hols [59]. The trade-off between centralized network-wide state visibility and scalability of the distributed control plane has attracted researchers to propose different controller state distribution techniques for better scalability and performance under large-scale and dynamic network traffic and topology conditions.

Hu *et al.* [60] provided a survey on the recent solutions for maintaining a consistent global network state view among multi-controllers. The literature classified the existing solutions into two aspects: 1) consistency of control state; 2) consistency of control strategy. However, in this survey, we aim at presenting the research efforts toward re-engineering the inter-controller traffic to prevent controllers from being overloaded and therefore enhancing the network scalability. Based on how the global network state view is maintained and distributed among controllers, we classify the proposed distributed global network state view into three main schemes; flat controller state distribution, hierarchical controller state distribution, and hybrid controller state distribution as illustrated in Figure 7.

A Comparison of the different controller state distribution mechanisms is also provided in Table 2.

A. FLAT CONTROLLER STATE DISTRIBUTION

Controllers of this scheme are horizontally distributed and share their state view using a distributed data store, hash

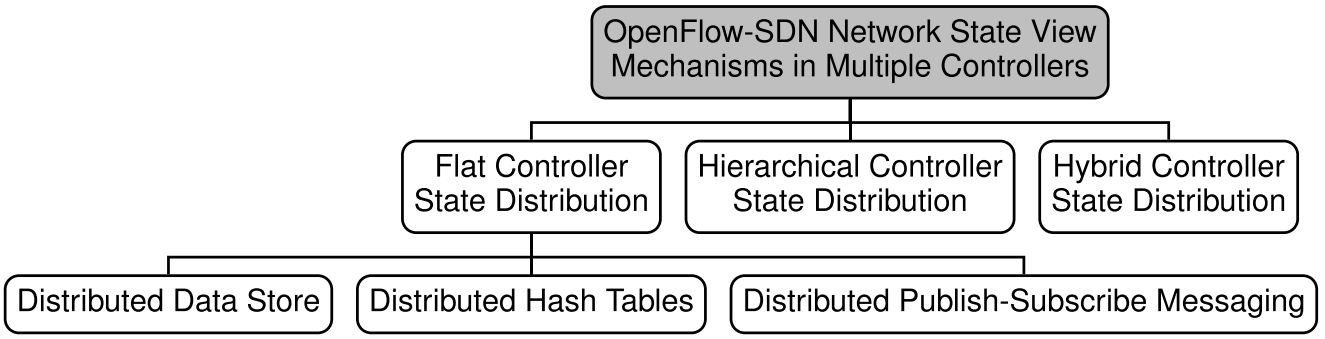

FIGURE 7. OpenFlow-SDN network state view mechanisms in multiple controllers.

tables or publish-subscribe messaging to build a consistent and centralized global view of the network.

1) DISTRIBUTED DATA STORE

Controllers store their network state view in a flat-distributed data store (e.g., Network Information Base (NIB), Stores, sharding). Each controller in this scheme synchronizes its state domain with other controllers to realize centralized visibility over the network. The authors in [61] propose two periodic synchronization schemes, i.e., Link Balance Controller (LBC) and Separate State Link Balancer Controller (SSLBC) to minimize the maximum inter-controllers link utilization. However, the frequent periodic synchronization may result in synchronization traffic overhead in the distributed controllers. Moreover, these schemes are classified under eventual consistency (EC) model [63], which can temporarily introduce state inconsistency, and therefore causes routing problems, such as forwarding loops and black holes, due to the inconsistency in the interval between two consecutive synchronizations. To overcome these issues, the authors in [66] propose an event-based controller synchronization mechanism so-called Load Variance-based Synchronization (LVS) that synchronizes controllers only when a load of a specific controller or domain exceeds a certain threshold. The proposed mechanism reduces the synchronization overhead as possible by updating all controllers with the crucial variance of network status that could lead to forwarding loops.

A controller as ONOS [12] uses the RAFT [67] consensus algorithm to maintain strong consistency. Each controller replica is assigned a follower, leader or candidate role. Following a committed state-update at the leader controller, the update is propagated to the follower controllers only after half of them have agreed on the update. Unlike the eventually consistent model, the strong consistency comes at the cost of increasing controllers' response time and lowering their availability. ONOS [12] is a logically centralized open-source controller which follows in the footsteps of ONIX [10] controller. It builds on the open-source single-instance Floodlight [40] SDN controller. Each ONOS instance is responsible for propagating the network state of its data-plane view with the global network view. Controllers are instantiated as the capacity of data plane grows. Each switch in ONOS requires to be connected to multiple controller instances to maintain a fault tolerance, but only one instance is selected as a master. If a master instance fails, a new instance is selected from the remaining pool of instances to be the new master. Multiple instances of RAFT protocol is running simultaneously to maintain strong consistency among the controllers. To improve scalability, the controllers' network state is partitioned into shards of distributed *stores* data structure, where each shard is managed by a different RAFT instance and shared among not more than 3 controllers. The research work in [59] studied the inter-controller traffic and realized that the significant source of traffic overhead is mainly from the consensus protocols. The study developed some empirical models to quantify the traffic exchanged among the controllers, depending on the considered shared data structures.

Adaptive consistency is also proposed in some recent works as in [62], [63] and [68]. In this scheme of consistency, the periodic synchronization is tuned according to the current network state to achieve a certain consistency level. Changing the controller consistency level on-the-fly can maintain a scalable system that sacrifices application optimality for less synchronization overhead. Adaptive consistency can achieve consistency and availability among controller with lower synchronization traffic overhead, and therefore enhances the network scalability.

2) DISTRIBUTED HASH TABLES

Distributed Hash Table (DHT) is a type of decentralized distribution that provides a lookup similar hash table of key-value pairs to partition data among distributed nodes. In this scheme, DHT is responsible for distributing the storing network state views among controllers. Each distributed controller has a unique global identification (GUID) and a hash table that contain a key and value pairs of its local link-state views. Retrieving a certain link state view is performed by sending a request message to the controller of an index (GUID) that hosts the key resulted from a hash function of that value. DHT based distribution of the network-wide

TABLE 2. A comparison of proposed openFlow-SDN controllers state distribution mechanisms.

Proposed Approach	Mode	Controller State Synchro-nization	Consistency	Method	Consensus algorithm	Pros.	Cons.
ONIX [10]	Flat	Periodic	Strong/ Eventual	Distributed Hash-tables	Paxos	-Allow applications to make their trade-offs among consistency, durability and scalability	
Levin et al. [61]	Flat	Periodic	Eventual	Transactional Database on NIB		-High availability -Controllers load balancing	-Inconsistency intervals which can lead to improper forwarding decisions
ONOS [12]	Flat	Periodic	Strong/ Eventual	Distributed stores data structure (ZooKeeper)	RAFT	-Allow applications to make their trade-offs among consistency, durability and scalability	
[13]	Flat	Periodic	Strong	Publish-Subscribe	RAFT	-Fault tolerance -Loop-free forwarding	-High synchronization traffic -High response time
HyperFlow [44]	Flat	Event-driven	Eventual	Publish-Subscribe (WheelFS)		-Low communication overhead -Low response time -High availability	-Inconsistency intervals which can lead to improper forwarding decisions
Zhang et al. [62]	Flat	Periodic	Adaptive	Transactional Database on NIB		-Low communication overhead -Low response time	
Sakic et al. [63]	Flat	Periodic	Adaptive	Transactional Database on NIB		-Low communication overhead -Low response time	
DISCO. [11]	Flat	Event-driven	Eventual	Publish-Subscribe		-Low communication overhead -Low response time -High availability	-Publishers and subscribers are tightly coupled to a broker
PLEROMA [64]	Flat	Event-driven	Strong	Publish-Subscribe		-Application-aware control distribution	-Publishers and subscribers are tightly coupled to a broker
ZeroSDN [65]	Flat	Event-driven	Eventual	Publish-Subscribe		-Low communication overhead -Low response time -High availability	-Push control logic to the data plane and limit fine-grained control over the network
Guo et al. [66]	Flat	Event-driven	Strong	Load variance-based synchronization		-Controllers load balancing -Low communication overhead -Loop-free forwarding	-Non-optimal routing decisions in highly dynamic networks -Latency has not been evaluated
Kandoo [14]	Hierarchical	Periodic	Eventual	Hierarchical controller state distribution		-Low communication overhead -High availability	-Lead to a path stretch problem -High response time
ORION [16], [48]	Hybrid	Periodic	Eventual	Hierarchical controller state distribution		-Low communication overhead -Low computational complexity -High availability	

state views among controller instances is used to avoid overwhelming controller resources due to replicating and updating network stat at all controller instances to scale to a vast network. However, it increases flow-rules placement latency as a result of inter-controller communications to access topology state for calculating the routing path of the control plane packet_in flow-rules placement requests.

ONIX [10] and Beehive [21] are examples of distributed controllers that use the DHT distribution primitive to obtain logically-centralized control over the data plane. ONIX has two types of consistency mechanisms for synchronizing network state updates among the controller instances: a replicated transactional database designed for ensuring strong consistency, and a distributed hash table for maintaining an eventual consistency. Each ONIX instance shares and disseminates its view of the underlying network state to other instances within the network domain. The management and control layer is implemented on top of ONIX's APIs and responsible for controlling the network behavior. ONIX offers general-purpose APIs for control applications while allowing them to make their trade-offs among consistency, durability, and scalability. The controller can actively cache its local view of the corresponding data plane elements while reactively request other remote controllers' views to maintain

fragmentation transparency over the data plane forwarding elements and ensure availability and scalability.

3) DISTRIBUTED PUBLISH-SUBSCRIBE MESSAGING

The publish-subscribe messaging pattern is a way of decoupling the information providers and consumers. The provider (publisher) disseminates data without knowing the consumer while the consumer (subscriber) registers its interest to the required data. Each controller can publish its local network state view to other controllers without the delay resulted from request-reply of the client-server communication model. This pattern can be used efficiently to maintain global network state view over the distributed controllers while guarantees better control plane scalability.

HyperFlow [44] is one of the early distributed event-based controllers for OpenFlow using the publish-subscribe messaging which implemented as an application for NOX controller [38]. HyperFlow uses WheelFS [69] distributed file system with a familiar POSIX interface which allows applications to adjust the trade-off between prompt visibility of updates from other sites and the ability for sites to operate independently despite failures and long delays. WheelFS allows these adjustments via semantic cues, which provide application control over consistency, failure handling, and file and replica placement. HyperFlow proposes to localize decision making to interconnecting independent distributed controllers that can serve data plane without contacting any remote controller. To reduce the controller response time and achieve better scalability, HyperFlow implements event-based publish-subscribe messaging to propagate and eventually replicate the controller's network view state among all distributed controllers. Every controller has a HyperFlow instance which selectively publishes the events that make a change to the state of the networks. HyperFlow controllers in a specific domain can get most of the updates of other domains from nearby controllers, thus minimizes the cross-site traffic required to propagate synchronization events.

Although, the HyperFlow controller passively publishes state to other controllers to significantly reduce the response time of contacting remote controllers, it does not efficiently reduce the amount of link capacity required for maintaining inter-controller consistency in a highly dynamic and scale-out networks and assumes that only a tiny fraction of network events cause changes to the network-wide view. Furthermore, it can not prevents the routing problems as forwarding loops caused by the temporary inconsistency during the passive propagation of synchronization traffic among controllers.

OpenDayLight [13] controller platform is a distributed controller that introduces modularity in implementing control functions by using Model-Driven Software Engineering (MDSE) specification. OpenDayLight went further beyond the basic premise of SDN and supported multiple southbound protocol plugins, services, and applications. Furthermore, it supports some programmability technologies and SDN platforms, including OpenFlow, OVSDB, NETCONF/YANG, and BGP, thus allows application developers to focus more on SDN APIs rather than underlying network communication protocols. OpenDayLight uses RAFT [67] consensus algorithm to maintain strong consistency. It also supports both request-reply and publish-subscribe communication patterns by implementing a Model-Driven SAL (MD-SAL) service bus in each controller. The request-reply pattern is implemented by RPC module while publish-subscribe functionality is provided by notification module.

DIstributed SDN COntrol (DISCO) [11] controller uses (topic-based) publish-subscribe messaging based on the Advanced Message Queuing Protocol (AMQP) to control multi-domain SDN. DISCO is a flat distributed controller implemented on top of Floodlight [40] controller and provides a lightweight and highly manageable inter-controller channel to let all controllers share their link-state views. DISCO does not impose a strong consistent network-wide state in all controllers and provides a distributed control plane for WAN and constrained networks based on a message-oriented communication bus. DISCO controller consists of two main parts (intra-domain and inter-domain) with different functionalities to reduce the overall inter-controller consistency traffic using topic filtering to adapt to the heterogeneous network topologies dynamically. Intra-domain part gathers the main functionalities of the controller while inter-domain part manages the communication with other DISCO controllers (e.g., reservation, topology state modifications, monitoring).

PLEROMA [64] is another example of a flat distributed controller that uses the content-based publish-subscribe communication model to distribute the network-wide state view and obtain logically-centralized control over the data plane. Two main components (dispatcher and configurator) in PLEROMA are responsible for handling the events among publishers and subscribers. The dispatcher (broker) component is responsible for collecting control requests and events from publishers and subscribers. On the other hand, the configuration components are responsible for processing these requests and performing network updates accordingly. However, using brokers to save bandwidth in a publish-subscribe model can impose a significant delay by lengthening the end-to-end path with a detour to the brokers and a processing delay for matching events against filters' rules.

ZeroSDN [65] splits control logic into lightweight control modules so-called *controllers*. The lightweight controllers allow for pushing control logic onto switches and enable local processing of data plane events to minimize control latency and communication overhead. ZeroSDN uses a publish-subscribe messaging to implement message bus which enables event-based communication among decoupling controllers and data (forwarding) elements. The network-wide state is obtained by using the topology controllers which subscribe to both SwitchRegistry and LinkDiscovery events of certain partitioning topology groups to obtain global network topology knowledge.

Like Hash-based distribution, publish-subscribe based distribution of global network state view reduces the overhead of maintaining replica in each controller and therefore allows control plane to scale. However, it still requires some time to access topology state for calculating the routing path of the control plane reactive flow-rules placement requests. Controllers, on the other hand, can efficiently utilize its insight over the data plane to predict the expected network flows, and thereby subscribe proactively to the most critical and required application-based network state. Thus, significantly reduce the overall amount of inter-controller consistency traffic required for maintaining network-wide state visibility and reduce the link utilization in the large-scale and dynamic networks environments as IoT.

B. HIERARCHICAL CONTROLLER STATE DISTRIBUTION

Reducing the overhead of frequent events on the control plane is essential for realizing an agile and scalable SDN network. The flat-distributed control plane architecture cannot solve the super-linear computational complexity growth of the control plane when SDN network scales to large size [48]. As a result, some research works focus on how to reduce the amount of control plane traffic by vertically distribute the control plane as in Kandoo [14], Logical Xbar [15], and ORION [16]. Controllers on this architecture are portioned among multiple layers, typically two or three layers. This design gives a simpler approach to managing controllers where controllers have different responsibilities and can make decisions based on their level and view on the network [70]. The low-layer controllers are responsible for only their regional data plane, while top-layer controllers offer a logically centralized and global view control over the whole network.

Kandoo [14] as an instance proposes a two-layer structure of control plane. The bottom layer controllers are not interconnected and only know its local network state view. The top layer is logically centralized controllers which maintain a global network state view over the whole network. Controllers at the bottom layer can handle small and frequent flows that can be processed using the local state knowledge of the controller to reduce the load on the root controller effectively. Top layer's controller, on the other hand, takes charge of the large volume streams (elephant flows) that requires network-wide state for optimal routing path. Root controller also acts as a mediator between bottom layer controllers. Kandoo limits the overhead of consistency traffic on the control plane to achieve synchronization among controllers and relieve the load on the top layer. However, contacting the upper layer to calculate the optimal forwarding rules of any new flow-rules placement request brings a path stretch problem and increases the controller response time.

C. HYBRID NETWORK STATE DISTRIBUTION

In the hierarchically distributed control plane architecture, if the data plane forwarding request requires non-regional routing information, it will be directed to the upper layer domain controller. As such, the routing path increases by the number of hops to the upper layer which brings a path stretch problem and increases the controller response time. As a result, some research work proposes a hybrid hierarchical control plane architecture to resolve the path stretch problem and decrease the controller's response time. ORION [16], [48] as instant, addresses this issue by proposing a hybrid hierarchical control plane that acts as the flat-distributed control plane architecture while hierarchically distribute the controllers. The control plane has two layers: the bottom layer includes local area controllers which are responsible of collecting physical device and link information, dealing with intra-area requests and updates, as well as abstracting the network view and sending it to the upper management layer. The upper layer, on the other hand, contains the domain controllers which maintain the global network view for the bottom layer. ORION has a routing module which can effectively reduce the path stretch problem of the hierarchical structure. The domain controller can calculate the shortest path by collecting the intra-area hops from the inner switch to all edge switches which is sending by area controllers, and adds the inter-area hops and the intra-area hops together. Thus, abstracting views from the area to the domain layer can reduce the problem of computational complexity in large-scale networks. Although ORION can effectively reduce the path stretch problem of the hierarchical structure, it still requires to contact the domain controllers when the destination address of the flow-rules placement request is out of the area. Moreover, the vertical communication between the domain and area controllers is established via a request/replay TCP which add extra propagation latency. The bottom layer needs to be exposed to the network-wide state by the upper layer through a messaging bus to reduce the controller response time. Thus, calculating the routing path of new flow can be done in the area layer and no need to send packets to the upper domain layer.

V. OPENFLOW-SDN LINK STATE DISCOVERY

In the OpenFlow-SDN, the data plane forwarding elements are meant to be simple forwarding devices. The implementation of networking logic such as routing or link discovery services is the control plane responsibility. Hence, there is no official discovery protocol standard for the OpenFlow-SDN, and most of the current OpenFlow-based controllers implement the Link Layer Discovery Protocol (LLDP) standard [71]. LLDP is a vendor-neutral L2 single-hop protocol that allows IEEE 802 local area network devices to advertise their identity, capabilities and direct connected neighbors. Each LLDP discovery message encapsulated in an Ethernet frame with an EtherType field (sets to 0x88cc) while each frame contains one data unit (LLDPUD) which consists of a sequence of type-length-value (TLV) variables as illustrated in Figure 8. LLDP stores the gathered information in the forwarding device's management information database (MIB) which can then be queried when crawling the network's nodes to retrieve the network state topology

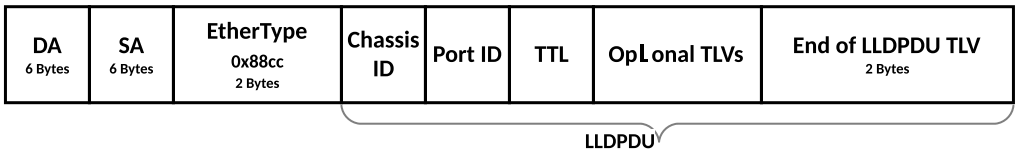

FIGURE 8. LLDP frame.

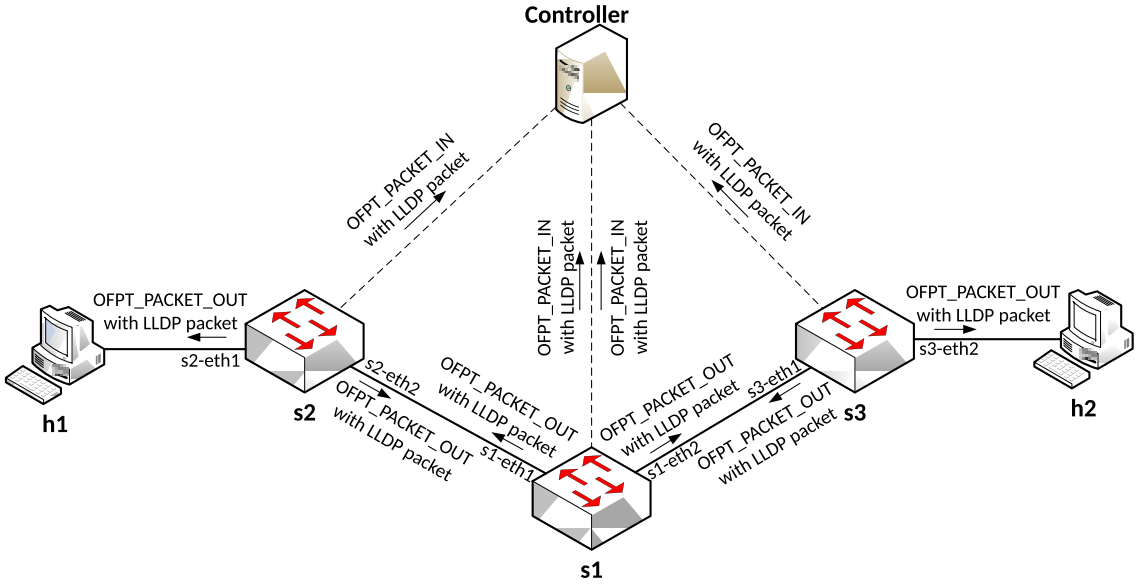

FIGURE 9. Open flow discovery protocol.

using a network management protocol like SNMP. NOX [38] controller is the first SDN controller that leverages the Link Layer Discovery Protocol (LLDP) in its implementation with minor modification to perform the process of discovering the network topology which is then known as OpenFlow Discovery Protocol (OFDP). Unlike the single-hop advertising only feature of LLDP, OFDP is request-reply discovery protocol that can receive the gathered discovery information by sending a *packet_in* message to the controller.

In OFDP, the controller initiates the discovery process by sending an LLDP discovery advertisement which encapsulated in a *packet_out* message to the directly connected forwarding devices using the OpenFlow Multicast address. When an OpenFlow/non-OpenFlow forwarding device receives the advertisement message, it floods all of its ports with the received LLDP discovery advertisement, and only the OpenFlow-enabled one updates its OFDP table. To explain the OFDP discovery process, Figure 9 shows an SDN topology of one controller and three linearly connected forwarding nodes $S1$, $S2$ and $S3$. The discovery process can be separated into two phases: the handshaking and configuration phase and the link-layer discovery phase as shown in Figure 10. In the first phase, each data forwarding node initially sends a handshaking hello messages *ofpt_hello* to the assigned remote controller. The controller responds with a message *ofpt_feature_request*, requesting more information. As a response, each data forwarding node sends a message *ofpt_feature_reply* to provide the controller with the relevant discovery parameter such as node ID and active ports with their respective MAC associates. The controller will then send a *ofpt_set_config* to install the rules of forwarding LLDP encapsulated packet_in/out packets. In the next phase, the controller sends an LLDP packet encapsulated in a *packet_out* (OFDP) messages to every connected port or forwarding node (as optimized in OFDPv2 [49]) immediately after handshaking. The advertising message will then be flooded to all ports of the adjacent nodes using the OpenFlow Multicast address excluding the ingress port (controller port). As such, each forwarding node can advertise itself to the adjacent nodes.

LLDP is a one-way single-hope advertising protocol, so when a forwarding node receives a forwarded packet_out message by a port that is not the controller port, the executed table-miss instructions will encapsulate the packet within a *packet_in* and send it to the controller including the discovery information of the source and adjacent nodes. After the controller receives all the packet_in messages, it will have complete link information between each connected OpenFlow-based forwarding node. The discovery process is performed periodically every amount of time [72].

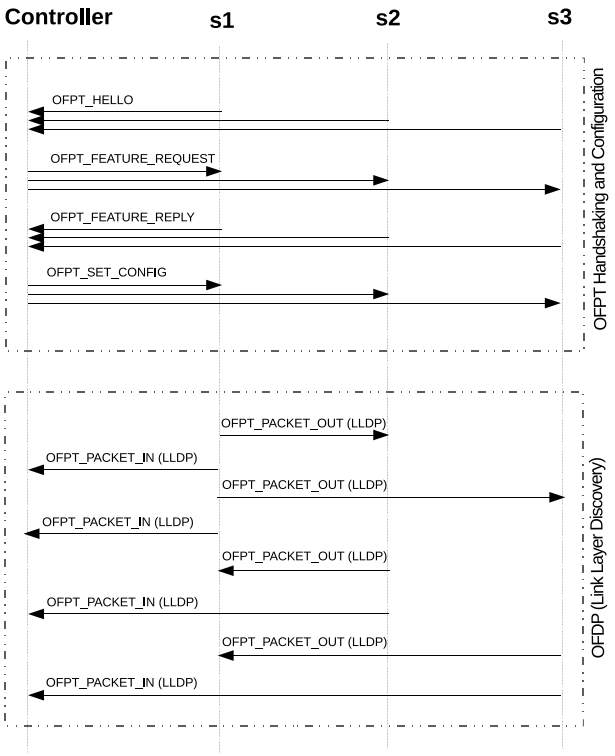

FIGURE 10. Open flow discovery protocol flow sequence.

According to [49], the controller requires to send an LLDP packet_out for each port in each forwarding node, and in turn, an LLDP packet_in will be sent to the controller form each forwarding node. The total number of LLDP packet_in and packet_out messages can be calculated using Eq. 1, where S refers to the forwarding nodes, L refers to the links between all forwarding nodes and P refers to the ports.

$$TOTAL_{(packet\_in/out)} = 2L.\sum_{i=1}^{S} P_i \tag{1}$$

In a large-scale network of highly dynamic topology, keeping up-to-date visibility of the topology is critical for performing optimal routing decisions by the controller. However, this large number of regular discovery traffic can lower the network throughput and hinder OpenFlow-based networks performance and scalability. Furthermore, enabling a global centralized control over a flow-based network requires that the controller has to have an up-to-date real-time view of the network state to act as a centralized controller and efficiently serve the data plane forwarding requests and network applications. The entire control plane management procedure is substantially affected by how efficiently and timely it can discover data plane forwarding nodes and links to maintain a centralized global view of the network's topology [50]. Link state discovery is crucial for network services that require real-time network state view. It is also essential for performing traffic engineering in a highly dynamic network of traffic and topology conditions. Some researchers focus on how to enhance the OpenFlow discovery protocol to improve the overall network performance and scalability. In this survey, we classify the proposed solutions into three schemes: LLDP-based Link State Discovery, Tree Exploration Link State Discovery and Layer-2 based Link State Discovery as illustrated in Figure 11. A Comparison of the different link-state discovery mechanisms is also provided in Table 3.

A. LLDP-BASED LINK STATE DISCOVERY

In this scheme, researchers focus on how to enhance the de facto OpenFlow discovery protocol (OFDP) to reduce the overhead of discovery traffic and improve the SDN scalability and performance.

The authors in [49] propose a modification to the de facto implementation of OFDP, called OFDPv2 to reduce the overhead of OFDP discovery protocol by reducing the number of *packet_out* messages that are required to be sent by the controller. The idea is to send only one LLDP *packet_out* message to the forwarding node and provides instructions to the forwarding node to forward it via its ports, after adding the *Port ID* TLV to allow the adjacent egress node to identify the source port. This proposed solution uses the OpenFlow feature of rewriting packet headers to rewrite the MAC address of forwarded LLDP packet. Another research work in [73] proposes to include also hosts in the discovery to reduce the ARP flooding when hosts initially generate traffic.

The OpenFlow-based discovery process is periodically triggered to provide the controller with the current global topology view. However, this may introduce unnecessary, redundant discovery traffic to the controller. As a result, research works as [50], [74] propose event-driven discovery mechanisms to resolve this issue. In [74], the authors propose a secure and efficient discovery protocol called sOFTDP which implemented in the Floodlight controller [40]. sOFTDP enables forwarding nodes to detect link events and asynchronously notify the controller autonomously. The controller then keep listening only for link event notifications from the data forwarding nodes to make topology updates.

Unlike OFDP and OFDPv2, the authors in [50] emphasize that all information required by the controller to create the topology map can be automatically extracted from the network devices using the existing protocol without the need to use the modified OpenFlow-based version like OFDP. Aiming at obtaining a discovery mechanism capable of fetching topology information from SDN and non-SDN devices, this work proposes to use communally existing protocols like ARP and LLDP without any modification. It proposes an event-based listening mechanism in each forwarding node's port to send information to the controller whenever traffic from a predefined protocol was detected. This way, the controller can get information about the forwarding node and its neighbor nodes. Using only the existing protocols to discover the topology in SDN-based networks can efficiently reduce the overhead of sending control traffic and solves the topology discovery problem in a hybrid SDN network. However, it limits the controller ability to get statistical information about the discovery traffic in the data plane and requires to

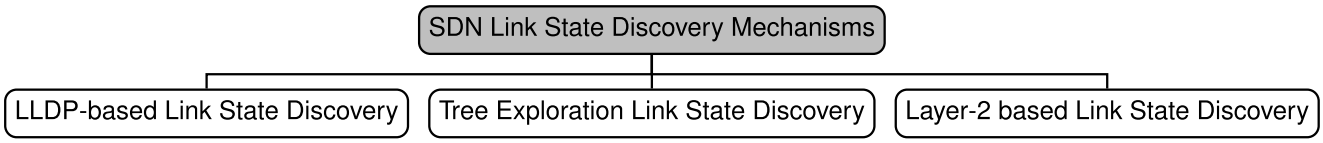

FIGURE 11. SDN link state discovery mechanisms.

TABLE 3. Comparison of the proposed SDN link state discovery mechanisms.

Proposed Approach	Periodic/ Event-driven	Method	Discovery Protocol	Hybrid SDN	Objectives	Cons.
OFDPv2 [49]	Periodic	LLDP-based	OFDP-LLDP		Reduces the overhead of sending LLDP packet_out to every port in the forwarding node to only one packet	Periodic discovery traffic overhead and does not support hybrid-SDN of non-OpenFlow forwarding elements
Lopez et al. [73]	Periodic	LLDP-based	OFDP-LLDP		Includes hosts in the discovery process of OFDP to reduce ARP flooding	Periodic discovery traffic overhead and does not support hybrid-SDN of non-OpenFlow forwarding elements
sOFTDP [74]	Event-driven	LLDP-based	OFDP-LLDP		Eliminates the overhead of periodically sending unnecessary, redundant discovery traffic	Does not support hybrid-SDN of non-OpenFlow forwarding elements
Tarnaras et al. [50]	Event-driven	LLDP-based	LLDP and ARPs	✓	Eliminates the overhead of periodically sending unnecessary, redundant discovery traffic	Limits the controller ability to get statistical information about the discovery traffic in the data plane and requires to modify the forwarding node so it can trigger asynchronous events to the controller
TEDP [75]	Periodic	Tree Exploration	TEDP	✓	Reduces the traffic overhead of one-hop point-to-point LLDP-based discovery process by sending only one prob frame from the controller	May introduce a delay in updating the controller with up-to-date topology view especially in the context of highly dynamic large-scale networks
SD-TDP [76]	Periodic	Layer-2 based	SD-TDP	✓	Discovery traffic is aggregated and sent by only few forwarding nodes which significantly reduces the number of messages exchange	Data plane forwarding element needs to be modified to support the proposed agent-based mechanism. Moreover, aggregating the discovery topology information can lead to delay constraints
eTDP [77]	Periodic	Layer-2 based	eTDP	✓	Layer-2 discovery without the need for previous network configurations or controller knowledge of the network	Data plane forwarding elements need to be modified to support the proposed agent-based mechanism

modify the forwarding node so it can trigger asynchronous events to the controller. Furthermore, the proposed method is only implemented and tested on ForCES, and need to be tested on OpenFlow too.

B. TREE EXPLORATION LINK STATE DISCOVERY

Another way of discovering SDN topology proposes in [75] so-called tree exploration discovery protocol (TEDP), which reduces the traffic overhead of one-hope point-to-point LLDP-based discovery process. Instead of sending and receiving discovery messages among neighbors, TEDP proposes to send only one single probe frame from the controller, which then floods the network and explores the whole network topology. Unlike OFDP, TEDP collects the topology information at each hop and sends it directly to the controller. As such, the controller can find the optimal path between nodes without additional traffic cost. Although sending one single flooded discovery advertisement reduces the number of messages that need to be sent by the controller to only one, it may introduce a delay in updating the controller with up-to-date topology view especially in the context of highly dynamic large-scale networks.

C. LAYER-2 BASED LINK STATE DISCOVERY

More optimization for OFDP is also proposed by SD-TDP [76], in which discovery traffic is sent only by few forwarding nodes in a hierarchical SDN network architecture. SD-TDP reduces the number of messages exchange in OFDP by proposing an agent-based layer-2 only topology discovery protocol (SD-TDP) that divides the discovery process into phases and distributes hierarchically the discovery functions between the network nodes. Thus, allows for obtaining the network graph as quickly as possible without incurring scalability issue. SD-TDP proposes to select nodes for aggregating the topology information and send it to the controller. The drawback of this solution is that each forwarding node has

to support the proposed algorithm through an agent which follow a hybrid non-standard SDN. Furthermore, the aggregating of discovery topology information can lead to delay constraints.

In addition to the hierarchical distributed discovery, the authors in [77] state that the discovery process in LLDP-based discovery protocols starts after handshaking process in which controllers have prior knowledge of IP addresses and the list of active ports of each SDN forwarding element. As a result, it proposes a layer-2 discovery protocol so-called Enhanced Topology Discovery Protocol (eTDP) that hierarchically distributes the discovery functions among forwarding elements. The discovery process in eTDP can start without the need for previous network configurations or controller knowledge of the network. However, eTDP requires that every forwarding element has an eTDP agent, which need modification on both OpenFlow-based/non-OpenFlow-based forwarding elements. Moreover, eTDP is not a listening event-based protocol and depends on the periodic exchange of topoReply messages to inform the controller about any link failures or port-down. The periodic exchange of discovery messages can lead to unnecessary bandwidth wastage and control traffic overhead especially when the network topology changes rarely.

Previous literature does not give a robust study and analysis on the effect of discovery protocol in the context of large-scale SDN-based networks of dynamic traffic and topology conditions. Discovery process is either statically configured to be repeated periodically to keep up the controller up-to-date with the global topology view or reactively configured to update the controller when receiving links change events. However, the statically configured discovery process may result in unnecessary discovery traffic in long-time non-changing topologies while reactive configuration needs to be fast enough to avoid packet loss or routing mistakes. Furthermore, most of the proposed OpenFlow-based discovery protocols are for specific networks type and cannot work in a heterogeneous network of both wired and wireless devices like IoT. Therefore, more research requires for proposing an efficient and dynamic discovery protocol for large-scale heterogeneous networks where topology and traffic dynamically change. The discovery process has to be triggered asynchronously in response to the reactive or predicted topology changes to save network resources and avoid sending redundant topology information to the controller.

VI. OPENFLOW-SDN FLOW RULES PLACEMENT

The controller involvement in response to every new unknown flow arrives at the forwarding node can be avoided by installing flow entries proactively. Rather than reacting to the arriving flow packets, the OpenFlow controller can proactively populate the flow entries of all traffic matches. The long non-prefix matching of flow entries can match the most granular route to a destination. As a result, all flows and actions can be predefined and installed to the data forwarding node in advance to the flow arrival and preventing any *Packet_In* flow-rules placement request to the controller. Such proactive action is suitable for simple network's topologies where flow tables memory size is fit with the global network state view. However, in large-scale networks, the TCAM-based flow tables can be overflowed by the high number of installed flow entries that match traffic between all ingress and egress ports. Although proactive flow-rules placement approach is preferable for achieving better performance, the flow control granularity is broken when new flows are forwarded without the controller awareness or when the flow entry compression is achieved based on the relationship among flow entries [78]. The flow granularity is essential to obtain better SDN controller's visibility over the network and therefore guarantee an optimal QoS provisioning and controlling by monitoring and collecting per-flow statistics. Meanwhile, allocating the pre-calculated forwarding rules before flow transmission without reacting to the dynamic changes can lead to routing problems like routing loops, black holes and dropping packets.

OpenFlow protocol holds only the recently arrived flow entries in the flow tables by configuring every flow entry with an idle timeout to cope with the limitation of the flow tables memory size. The flow entries with expired idle timeout are removed from the flow tables to offer space for the newly installed flow entries, and thereby avoid flow tables overflow. However, this mechanism increases the communication traffic between control and data planes due to the controller involvement in every new flow-rules placement process. An on-demand flow-rules placement request is created and sent as a *Packet_In* by the forwarding element to the controller every time a new unknown flow arrives with unmatched flow entry. The controller then responds with the new calculated forwarding rule and adds it to the flow tables of the directly connected forwarding nodes. Such on-demand flow-rules placement process can introduce a significant delay and controller traffic overload in the case of a high number of congested new flows, and consequently, increases the controller response time and decreases data plane throughput as the network scale grows dynamically [37].

This section provides a survey on the OpenFlow-based state-of-the-art research studies on how to reduce the amount of traffic between control and data planes, and efficiently utilize the data plane flow tables in response to the dynamic nature and resource limitations of the network.

Nguyen *et al.* [79] provided a survey on the proposed solutions toward data plane memory management and reducing signaling traffic to improve SDN scalability. Compared to this work, we focus on the state-of-the-art proposed mechanisms toward reducing the traffic overhead between control and data planes caused by the reactive flow-rules placement, and classify them into five schemes: Control back to data plane, flow table entries reduction, per-flow source routing, adaptive flow entry's timeout and predictive flow rules placement. In addition, we cover also other OpenFlow SDN-scalability related issues. A taxonomy of the OpenFlow-SDN flow-rules

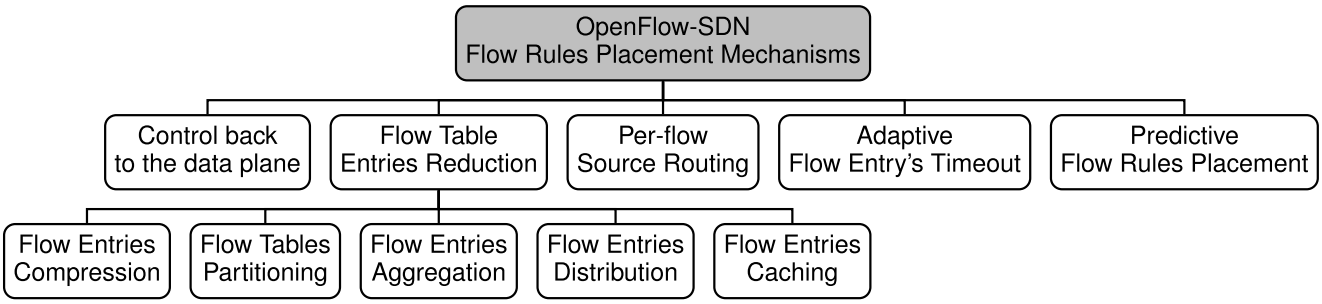

FIGURE 12. OpenFlow-SDN flow rules placement mechanisms.

TABLE 4. Comparison of the proposed openFlow-SDN control back to the data plane mechanisms.

Proposed Approach	Rules Placement	Method	Fine-grained Control	Objectives	Cons.
DevoFlow [80]	Reactive	Control back to data plane		Forwards short-lived flows (micro-flows) at the data plane without contacting the controller and only significant flows are reactively forwarded to the controller	Mandating changes of the switching ASIC and reactively detects elephant flows as they become significant
Song et al. [81]	Reactive	Control back to data plane	✓	Implements a controller-proxy that can delegate the control back to the data plane without sacrificing the advantages of the SDN centralized fine-grained control	Contradicts with the SDN standards and adds more complexity to the data plane when maintaining the network state view and synchronize it with the controller
ZeroSDN [65]	Reactive	Control back to data plane		Splits the control logic into control modules communicated through a message bus, allows network decisions on the local view and proposes a fast heuristic mechanism to quickly detect elephant flows	Contradicts with the SDN standards in making the networking devices as a simple decoupling forwarding nodes

placement mechanisms that have been proposed to enhance the network scalability and performance is also provided in Figure 12.

A. CONTROL BACK TO THE DATA PLANE

Some research works focus on how to gently break the coupling between the control and global visibility to the network state by reactively processing some traffic routing in the data plane without the control plane involvement, see Table 4.

DevoFlow [80] argues that fine-grained control and network-wide visibility of OpenFlow controller comes at the cost of involving the controller in every flow-rules placement and statistical gathering events. Therefore, such design of the OpenFlow model cannot meet the needs of high performance and scalable networks. DevoFlow proposes to bring back some of the control logic to the data forwarding elements, in a way that preserves central control and visibility over all significant flows only – aggregate flows that might become sufficiently intense (elephant flow), while limiting the load on the central controller. DevoFlow augments the action part of each OpenFlow packet with a Boolean CLOONE flag. If the flag value is clear, the switch follows the normal OpenFlow wildcard matching; otherwise, it locally clones the wildcard rule. The cloning process creates a new flow entry by replacing all the wildcard fields with values matching the (micro-flow) and inheriting other aspects from the original entry. As such, the switch locally routes short-lived flows (micro-flows) without contacting the controller and only significant flows are reactively forwarded to the controller. Thus, it reduces the load on the controller and improves the network scalability. The idea is to push as many decisions as possible to the data plane, but in a way that guarantees a simple and cost-effective hardware implementation. Although DevoFlow gently breaks the coupling between control and global visibility, it can maintain a useful amount of visibility without imposing unnecessary costs. However, it can not proactively detect the potential elephant flows, and instead, the controller detects elephant flows as they become significant.

Reactively processing some of the control logic in the data plane can significantly reduce the amount of traffic between control and data planes, and hence reduce the control traffic overhead on the controller. However, the amount of traffic among data plane elements to retrieve data-forwarding rules may introduce overhead in the data plane and can lead to potential routing loops and congestion due to non-optimal routing decisions and inefficient link utilization. Moreover, allowing data forwarding devices to decide on the routing of some flows contradicts with the main purpose of

SDN and comes at the cost of coarse-grained flow control. Hence, each packet may match multiple rules which consequently reduces the controller ability to implement fine-grained flow-level policies such as multipathing and prevents forwarding decisions from being calculated based on more information than just destination addresses.

Motivated by these drawbacks, Song *et al.* [81] propose a flow-rules placement mechanism so-called ''controller-proxy'' that can delegate the control back to the data plane without sacrificing the advantages of the SDN centralized fine-grained control. Although the data forwarding elements in this flow-rules placement mechanism can handle some event-processing logic, the control plane still maintains high visibility over the whole network by communicating with a proxy executor via a proxy path to synchronize the network state. The idea is to send an assist request to the hotspot data forwarding elements whenever traffic spikes occur to delegate the event-processing logic for *packet_in* events. The proxy executor in a hotspot data forwarding element can handle the new flows directly instead of sending a *packet_in* to the controller. Meanwhile, the packets from non-hotspots data forwarding elements are still handled by the controller. The proxy executor is periodically synchronizing the topology view with the controller so it can route the flows on time. The delegation process of the controller-proxy mechanism can improve the timeliness and scalability of the forwarding processes, but it pushes the overhead and complexity to the data plane. The amount of space required for saving the network view in the data forwarding element to maintain a fine-grained flow control consumes more memory and computational resources which can overload the data plane.

ZeroSDN [65] is another research work that supports the idea of bringing some of the control logic back to the data plane. ZeroSDN argues that a highly flexible SDN distributed architecture has to allow the full spectrum of distribution, from fully centralized to fully distributed architecture by including data forwarding nodes in the control distribution and allow network decisions on the local view. To minimize control latency, ZeroSDN focuses on the network timeliness by processing control decisions locally as possible while leveraging the logically centralized global network state view to improve the decision quality. Due to the nature of the local network state data of being the most recent and consistent, ZeroSDN, applies a fast heuristic mechanism to quickly decide whether the control event has to be processed locally or propagated to the middleware bus to be processed by remote entities in the control plane.

B. FLOW TABLE ENTRIES REDUCTION

To cope with the limitation of TCAM space and proactively install flow entries, some researchers focus on how to efficiently reduce the amount of memory space required to store flow entries by either compressing, aggregating, distributing or caching flow entries, see Table 5.

1) FLOW ENTRIES COMPRESSION

Some research works focus on how to efficiently compress the flow entries to reduce the number of flow requests arrived at the controller. Source Flow [82] is one of the early research works that propose to reduce the number of flow entries on core nodes without changing the granularity of flows. Source Flow proposes to remove the redundancy of flow matching rules for a selected routing path on all intermediate nodes. The idea is to embed the actions for all intermediate nodes as a form of a list into a user packet and store the forwarding rule actions as a pointer to an action table that stores the actual actions in the forwarding node. As such Source Flow can save the memory space to save redundant flow matching rules. However, the number of flow entries on edge nodes are not reduced and requires to modify OpenFlow to implement action table in the same way as a flow table.

The authors in [83] propose a compression mechanism called Compact TCAM which encodes the flow entry matching header while augmenting the action part of the rule with a boolean COMPACT flag. It proposes the use of shorter tags for identifying flows to optimize the TCAM space. The controller encodes the information of each unique flow entry as a numeric identifier so-called Flow-ID. The controller responds to each new flow request with a message consists of the flow ID and actions. Only in the egress switch, the encoded matching header is decoded back to save more TCAM space along the ingress and intermediate switches, and therefore allows storing more control flow entries. The packet's standard actions are executed if the COMPACT flag value is set to clear; otherwise, the packet's operations corresponding to the Flow-ID are executed. Compact TCAM needs to upgrade the OpenFlow standard flow tables to implement the Flow-ID table. The drawback of the compression mechanism is that it prevents the flow granularity along the forwarding path and therefore it cannot guarantee a fine-grained control over the network traffic.

2) FLOW TABLES PARTITIONING

In traditional IP routing tables, each routing entry mainly consists of destination IP, gateway IP and interface while the forwarding process requires only 32 bits (IPv4) or 128 bits (IPv6) of matching destination IP to forward the packet. However, in the OpenFlow protocol, each flow routing entry occupies 40 field tuples of 1227 bits memory storage while many of these fields are optional and empty, resulting in memory space wastage. Hence, compressing the match header fields is not enough to reduce the amount of memory space required to store flow entries. Accordingly, the authors in [78] propose a Heuristic Storage Space Optimization algorithm for Flow Tables (H-SOFT) to reduce the storage space required for each flow entry. H-SOFT partitions the flow table into simple files and stores them in multiple sub-flow tables, so it performs the compression by assigning each flow entry into those sub-flow tables. If the newly added flow entries exceed

TABLE 5. Comparison of the proposed openFlow-SDN flow table entries reduction mechanisms.

Proposed Approach	Rules Placement	Method	Fine-grained Control	Objectives	Cons.
Source Flow [82]	Reactive	Flow Entries Compression	✓	Removes redundancy of locating flow matching rules on all nodes on a selected path by embedding the actions of all intermediate nodes as a form of encoded list into a user packet and stores the actual actions in the forwarding node	The number of flow entries on edge nodes are not reduced and the proposed mechanism requires to modify OpenFlow to implement action table in the same way as a flow table
Compact TCAM [83]	Reactive	Flow Entries Compression		Reduces the amount of memory size that required to store flow entries by only compressing matching header	Requires to modify the OpenFlow protocol and affects the flow granularity
H-SOFT [78]	Reactive	Flow Tables Partitioning	✓	Releases the unused storage space in flow tables due to blank fields with much lower execution time than compression mechanism	Requires to modify the OpenFlow protocol
Braun et al. [84], Rifai et al. [85] and Zhu et al. [86]	Reactive	Flow Tables Aggregation		Aggregates the flow matching entries to more generic wildcards without the need to modify the OpenFlow protocol	Aggregating the fine-grained matching fields can lead to potential routing mistakes and affects the flow granularity
Luo et al. [87]	Reactive	Flow Entries Aggregation		Proposes a fast non-prefix aggregation mechanism to reduce the inconsistency interval between control and data plane caused by the long aggregation updating time	Aggregating the fine-grained matching fields can lead to potential routing mistakes and affects the flow granularity
FTRS [88]	Reactive	Flow Entries Aggregation	✓	Aggregating flow entries that are less important in the middle of the flow path while reserving the existence of flow entries at the edges to maintain fine-grained management without the need to modify the OpenFlow protocol	The flow tables at the edge forwarding nodes can be overflowed when a large number of new flows arrives
DIFANE [89]	Proactive	Flow Entries Distribution	✓	Distributes the pre-calculated forwarding rules and keeps the traffic in the data plane to avoid the reactive controller involvement for routing each miss-matching packets and enhance SDN scalability	Contradicts with the SDN standards, increases the propagation delay, and introduces a flow forwarding reliability and resiliency issues when authority switches fail
Palette [90]	Proactive	Flow Entries Distribution	✓	Decomposes the flow tables into small ones and distribute them across the data plane elements	Contradicts with the SDN standards and adds more traffic overhead in the data plane to maintain the distribution of forwarding rules
Kang et al. [91]	Proactive	Flow Entries Distribution	✓	Distributes the forwarding rules across an abstracted data plane layer so-called *one big switch* to maintain the TCAM rule-space constraints	Contradicts with the SDN standards and adds more traffic overhead in the data plane to maintain the distribution of forwarding rules
CacheFlow [92], [93]	Reactive	Flow Entries Caching	✓	Caches heavy-hitting rules in the TCAMS flow tables and the remaining rules in a software data structure	Breaks the long-chain rule matching dependencies leading to rule dependency issue
Sheu and Chuo [94]	Reactive	Flow Entries Caching	✓	Uses a (cover-set) based rules caching algorithm to solve the rule dependency problem and utilize the TCAM space more efficiently	Rules partitioning makes rule updates more difficult and challenging

the predefined threshold, H-SOFT dynamically adjusts the valid fields in each sub-flow table to optimize the storage space.

3) FLOW ENTRIES AGGREGATION

Although flow entries compression mechanism can efficiently optimize the size of the flow tables to accept more flow entries, the compression-decompression process requires high computational resources which add more overhead on the data plane especially in the condition of a large amount of aggregated traffic. Therefore, some research works focus on aggregating the original fine-grained TCAM flow entries into fewer coarse-grained ones with a larger matching range at the cost of losing some matching information. Unlike compression mechanism, flow entries aggregation mechanism can be implemented as a software plug-in on the OpenFlow controller and does not require any change in the OpenFlow protocol. Following the idea of prefix aggregation (summarization) in traditional IP routing to reduce the amount of routing tables, some research works such as [95], [84], [85] and [86] propose to aggregate the OpenFlow wildcard matching headers of the same action to allow installing more TCAMs flow entries.

Braun and Menth [84] as an instance, proposes to aggregate the flow matching entries to more generic wildcards using Espresso heuristic. Unlike IP prefix aggregation, flow entries contain a non-prefix matching field, which lengthens the flow tables updating time and hence increasing the flow-rules placement latency especially when the flow tables are frequently updated.

The aggregation process must not change the action part of any flow entry. As a result, motivated by Bit Weaving [96] TCAMs aggregation, the authors in [87] propose a non-prefix FFTA offline aggregation scheme that can aggregate 100 rule partition in just several milliseconds. It cuts the non-prefix matching fields to prefix permutable partitions and then aggregates each partition respectively. FFTA construct a Binary Search Tree (BST) of prefix partitions and then apply the Optimal Routing Table Constructor (ORTC) algorithm to omit the permutations and simplify the aggregation process. In the flow entries aggregation scheme, it is possible that a new unknown flow matches with an aggregated flow entry resulted in potential routing mistakes. As a result, Leng *et al.* [88] propose a Flow Table Reduction Scheme (FTRS) to solve this problem. FTRS pointed out that core switches are more likely to encounter flow table congestion than the boundary (edge) switches. Therefore, FTRS focuses on aggregating flow entries that are less important in the middle of the flow path while reserving the existence of flow entries at the edges to maintain fine-grained management. To achieve this, FTRS uses binary trie (prefix tree) to traverse a selected matching attributed from each flow entry (e.g., IP Address) into nodes and then reduce the number of flow entries by replacing the non-empty sub-trees with coarse-grained nodes.

Flow entries aggregation scheme is much faster than the compression scheme and consumes less computational resources. However, in dynamic and large-scale networks, the aggregation process may cause a routing delay because of the time requires to aggregate each new rule with all the existing ones. Furthermore, flow entries must share the same routing action to be eligible for aggregate. Hence, the high variety of the instructions field of flow entries can significantly reduce the number of possibly-aggregated flow entries.

4) FLOW ENTRIES DISTRIBUTION

Distributing the pre-calculated flow-rules among network forwarding nodes is another way to efficiently utilize flow table and reduce the communication traffic between the control and data planes, controller traffic overhead and hence improve the network scalability.

DIFANE [89] as an instance provides an architecture to distribute the pre-calculated forwarding rules and keep the traffic in the data plane to avoid the reactive controller involvement for routing each miss-matching packets. The controller first pre-calculates the forwarding rules and allocates them as even partitions to a *subset authority switches* of the existing ones. Upon receiving unknown new flow packet in the ingress switch, it will be encapsulated and sent to the appropriate authority switch based on the partition information. The authority switch de-encapsulates the packet and forwards it to the egress switch, meanwhile, sends feedback to the ingress switch to cache the flow entry to avoid long routing path for active flows. DIFANE provides a scalable solution that can efficiently reduce the control traffic overhead and eliminate the delay resulted from controller involvement in every new flow entry installation. However, it moves the complexity and overhead to the data plane by partitioning the flow entries in subtables distributed among selected authority switches. Furthermore, the authority switches play a main different rule in DIFANE which contradict with the main idea behind SDN in making the networking devices as a simple decoupling forwarding nodes. The coupling among the selected authority data forwarding nodes adds more complexity, unnecessary traffic on the data plane and increases the propagation delay. It also introduces a flow forwarding reliability and resiliency issues when authority switches fail.

To avoid the management and redirection overhead of packet forwarding at data plane in DIFANE, Palette [90] proposes a distributing framework for decomposing the flow tables into small ones and distribute them across the data plane elements. Kang *et al.* [91] on the other hand, proposes a rule replacement algorithm that distributes the forwarding rules across an abstracted data plane layer so-called *one big switch* to maintain the TCAM rule-space constraints. Rather than grappling with TCAM sizes, the control plane defines one big switch that manages the installation of rules on the data forwarding nodes.

Unlike other flow entries reduction mechanism, distributing flow entries among the data forwarding nodes reserves the fine-grained control policies. However, it adds more traffic overhead in the data plane to maintain the distribution of forwarding rules especially when the network topology changes dynamically.

5) FLOW ENTRIES CACHING

Flow entries caching is another solution to the limitation of TCAMS and the control channel traffic overhead. The idea is to cache heavy-hitting rules in the TCAMS flow tables and the remaining rules in a software data structure or software switches as part of the same hardware switch or on separate servers. Hence, give the controller the illusion of fast-forwarding/updating and large flow tables. In OpenFlow, if a flow packet matches multiple rules in a flow table, the data forwarding element will execute the actions of the highest priority matched rule. The problem with the rule caching scheme is that it breaks the long-chain rule matching dependencies leading to a rules dependency issue. In other words, if the high-priority rule of a certain flow is cached in the Non-TCAM flow table, the flow's arriving packet will incorrectly match the low-priority rule (because the priority is for rules cached in TCAMs). As such, the dependent rules of a TCAM-cached rule should also be cached to preserve the semantics of the fine-grained control policy. Meanwhile, the dependency checking mechanism has to be able to capture

all of the policies direct and indirect dependencies. Moreover, rule caching scheme should be adapted to the dynamic changes in the rule policies so it can update the cached rules in a proper time.

To resolve the rule dependency issue, CacheFlow [92], [93] proposes a hardware-software hybrid switch to cache the most popular rules in the available TCAMs flow tables while caching the remaining rules in a software switches. CacheFlow splices long dependency chains to cache small groups of rules while trying to preserve the semantics of network policies. To handle rule dependencies, CacheFlow represents the rule's priority as an annotated Directed Acyclic Graph (DAC) in which rules are incrementally added or removed. It also proposes a cache-replacement algorithm to decide which rules to place in the TCAMs.

Sheu and Chuo [94], on the other hand, proposes to use a cover-set approach to solve the rule dependency problem and utilize the TCAM space more efficiently. It proposes a (cover-set) based rules caching algorithm to cache the most frequent and important matched wildcard rules which have higher weight values. Unlike the standard cover-set caching algorithm, it calculates the accumulated contribution value of a set of rules instead of the individual contribution value of a rule. Moreover, it proposes a rule cache replacement (RCR) algorithm to increase the cache hit ratio. Once cache miss occurs, the RCR algorithm would replace cached victim rules with the cache miss rule to keep the rule in the TCAM. Although a well-designed rule cache replacement algorithm can resolve rules dependency problem, the rules partition makes rule updates more difficult and challenging, especially when rules change dynamically according to evolving network states.

C. PER-FLOW SOURCE ROUTING

The per-hop configuration-based forwarding in OpenFlow requires to generate multiple flow entries along a route to forward a single flow, resulting in a huge redundancy in flow tables. Hence, the flow tables space efficiency and management usually coupled with how the forwarding method is efficient. Instead of keeping the redundant forwarding rules in all the forwarding nodes along the route, some research studies propose to use source routing to make packet forwarding obeys the forwarding labels/instructions provided by the packet, see Table 6.

Having the global network state knowledge, the SDN controller can install the routing path information proactively to the ingress nodes. Thus the forwarding rule of the forwarded packet can be inspected in each node without the need to reactively contact the controller or proactively install the corresponding forwarding rules in the intermediate forwarding nodes. According to [110], source routing has been designed to be highly scalable, where the scaling capability of source routing has been tested successfully on a use case of 600,000 nodes and 300 millions of endpoints. Unlike traditional per-flow routing, no direct interaction is required between the SDN controller and each node along the route path. Furthermore, the controller can re-send the packet without the need for installing the forwarding rules in the related forwarding nodes [108].

The authors in [97], [98] propose to utilize source routing method to reduce the distribution of network state in all the data forwarding nodes along the routing path for improving the controller's scalability and network coverage time in an SD-WAN production deployment. It also proposes to change the output interface number in the path packet header with the input interface number at each intermediate node for maintaining the packet reverse path without sending a routing request to the controller. Although source routing can significantly enhance the SDN network scalability and performance, the controller involvement in calculating a new routing path is still necessary when a routing link fails. To avoid the consequent routing latency, a research study in SlickFlow [99] proposes a resilient source routing mechanism that combines the source routing with alternative path information carried in the packet header.

Some other research studies propose to encapsulate the arrived flow packet in the ingress port with multiple MPLS labels indicating the forwarding port numbers of other forwarding nodes on its route. Intermediate forwarding nodes will use the MPLS label to forward the packet to the next hop and delete the used header. This process is repeated in every intermediate forwarding node until the packet reached the egress port of the destination forwarding node. Thus, the controller needs to only install one flow entry in the ingress forwarding node, and perform the matching process once. However, using MPLS labels to forward flow packets in OpenFlow-SDN significantly reduces the redundancy of installing flow entries and the latency of the flow matching process. However, the packet headers needed to encapsulate the MPLS route labels generate too much overhead on the edge forwarding nodes and their links especially when the controller is configured to install all matching flow entries proactively. Each MPLS label requires 32 bits, which will be worse in a long route of multiple MPLS labels. The longer the forwarding path is, the larger the overhead becomes.

To resolve this issue, some research studies such as JumpFlow [100], Arbitrary Jump Source Routing (AJSR) [103] and Kitsuwan *et al.* [104] propose more efficient bandwidth utilization MPLS-like source routing techniques. JumpFlow [100] proposes to use the available VLAN identifier (VID) of the packet header to carry the routing information. JumpFlow considers the constraints of the flow table space and proposes to partition the routing information and distribute them on a few selected contact forwarding nodes. However, due to the limitation of 12-bits VID filed space and the reactive flow entries placement, JumpFlow can only carry little routing information and therefore cannot work properly in SD-WAN and may lead to a scalability issue. Inspired by JumpFlow, the authors in [101]–[103] propose to divide the complete routing path of a particular flow into arbitrary length sections and distribute these sections at different selected forwarding nodes along the route. The authors

TABLE 6. Comparison of the proposed openFlow-SDN per-flow source routing mechanisms.

Proposed Approach	Rules Placement	Method	Fine-grained Control	Objectives	Cons.
Soliman et al. [97], [98] and SlickFlow [99]	Reactive	Source Routing		Utilizes source routing method to reduce the distribution of network state in all the data forwarding nodes along the routing path	Edge nodes form a bottleneck due to the extra header size that required to encapsulate a multi-hop route
JumpFlow [100]	Reactive	Source Routing		Uses the available VLAN identifier (VID) of the flow entry's packet header to carry the routing information, and partitions the routing information and distribute them on a few selected contact forwarding nodes	Can only carry little routing information and therefore cannot work properly in SD-WAN
KSGT [101], [102]	Reactive	Source Routing		Divides flows into clusters and greedily selects a small number of switches to install the MPLS-based flow entries	Carrying routing path in MPLS labels may still cause a bandwidth overhead especially in large-scale networks and need for signaling the forwarding labels to the nodes along the route path
AJSR [103]	Reactive	Source Routing		Achieves a trade-off between the control traffic overhead and the bandwidth overhead by dividing the complete routing path of a particular flow into arbitrary length sections and distributes these sections in different selected forwarding nodes along the route	Can induce a significant bandwidth overhead due to carrying each per-hop forwarding information in one MPLS label and need for signaling the forwarding labels to the nodes along the route path
HPF [104]	Hybrid	Source Routing		Reduces both the number of permanent flow entries and the number of configuration messages from the controller by dividing switches into several regions and uses only two MPLS tags to forward packets within each region	May lead to a reconfiguration overhead when topology changes dynamically
Carpa et al. [105], Davoli at el. [106] and Sgambelluri et al. [107]	Reactive	Segment Routing		Uses segment routing in SDN to utilize the network bandwidth more efficient than traditional MPLS-based source routing mechanisms	Deep segment routing list in the packet header can cause a bandwidth wastage
Lee et al. [108] and Sheu et al. [109]	Reactive	Segment Routing		Uses heuristic routing algorithm with bandwidth guarantee to achieve traffic load balancing to resolve the extra wastage of the network bandwidth due to the deep SR list in the packet header	

in [101], [102] propose a heuristic algorithm called K Similar Greedy Tree (KSGT) to intelligently select nodes to install the MPLS-based flow entries. KSGT divides flows into clusters and greedily selects a small number of switches to install entries. However, using MPLS labels to carry each per-hop forwarding information can induce a significant bandwidth overhead. As a result, the authors in [103] propose a forwarding scheme called Arbitrary Jump Source Routing (AJSR) to achieve a trade-off between control traffic overhead and bandwidth overhead which can carry an arbitrary number of forwarding information and leverages MPLS labels to carry routing bath.

Segment routing (SR) [111] is another variant of source routing which has been proposed to provide flexibility and scalability to the traditional MPLS networking. SR can be implemented on top of either the MPLS or the IPv6. In MPLS-based SR, segment routing uses label forwarding but with no more additional protocol just extensions to Interior Gateway Protocols IGPs such as Intermediate System to Intermediate System (IS-IS) and Open Shortest Path First (OSPF). Segment routing uses the classical Dijkstra shortest path algorithm based routing protocols like IS-IS and OSPF to advertise the segment information and to compute the routing paths. However, in IPv6-based, segment routing uses IPv6 addresses to carry the segment list in an Extension Header called SR header (SRH) [112]. At the ingress node, SR encodes the routing path in the packet header as an ordered list of segments (stack of labels) to be executed on the subsequent nodes along the packet's route. There are two types of segments: nodal segment and adjacency segment. The nodal segment is globally significant and identifies the node and the prefix of its loopback interface. The adjacency segment is locally significant and identifies the local segment (e.g., switch port number) to a specific SR node [111]. Unlike MPLS, segment routing maintains per-flow state only at the ingress node. Thus, no need for signaling the forwarding labels to the nodes along the route path. Using segment routing in SDN-based networks can reduce the overhead of using Dijkstra-based routing algorithm to advertise the segment information because of the centralized view of global network topology that can be provided by the controller. Meanwhile, it can significantly minimize the need for keeping a large number of flow entries in the intermediate forwarding nodes. As a result, authors in [113] implement an SDN-based

segment routing to control the label stacks in a multi-layer network testbed for testing the scalability of the network. The results show that the deep label stacking can increase the average required time to configure the overall flow into 1ms without experiencing performance degradation. Segment routing attracts the researchers' interest to propose it in SDN because of its ability to utilize the network bandwidth more efficient than traditional source routing mechanisms.

The authors in [105] propose to improve the energy efficiency of large-scale backbone networks by using SDN and segment routing to dynamically adapt the number of powered-on/off links in response to the traffic load. They utilized segment routing to reduce the transmission overhead and the power of SDN centralized controller to selectively turn off a subset of links in response to the traffic load. The authors in [106] also propose an architecture that integrates SDN with segment routing. Trying to solve the Flow Rules Placement problem, they implemented a heuristic SR-based path assignment algorithm to find the minimum-length SR paths corresponding to each path. The scalability of the network when applying SR for SDN has been studied by [107]. An SDN-based SR implementation on a Multi-layer network is performed to demonstrate the scalability of the SDN network while performing dynamic traffic rerouting at the ingress node. The implementation uses OpenFlow 1.3 based RYU controller to control the SR segments configuration. It also relies on a Novel path computation algorithm to determine the routing path. The experimental results show that no packet loss was experienced during dynamic rerouting without the need for using signaling protocols.

Previous studies do not study the proper routing algorithms for SDN-based SR to reduce the extra cost of deep label stack in the packet header. In other words, if the count of path hops is long, it causes extra wastage of the network bandwidth due to the deep SR list in the packet header. As a result, the authors in [108] propose a heuristic routing algorithm with bandwidth guarantee. The proposed algorithm uses traffic engineering to select the proper routing path for achieving traffic load balancing among the network. The routing algorithm considers both the link's maximum residual bandwidth and minimum interface for selecting the routing (MIRA) methods to decide a routing plane for unicast communication in SDN. A multicast routing algorithm for SDN with SR is also proposed by [109] to serve the bandwidth requirements of multicast routing requests. The research work proposes a multicast heuristic routing algorithm with bandwidth guarantee to achieve traffic load balancing.

The authors in [114] went further and proposed an SDN-based SR on top of IPv6. Motivated by the centralized control of SDN, they propose an SDN architecture to control IPv6-based SR enabled networks. The segment list is carried in the segment routing header (SRH) as explained in IETF draft [112]. The proposed solution was applied in four different implementations of the southbound APIs: GRPC, REST and NETCONF, and remote command-line interface respectively. The research work also provides a comparison with OpenFlow-SDN solutions and represents that unlike IPv6, using OpenFlow as southbound API can lead to a lack of broad support and can easily transform in vendor lock-in.

Although source routing can significantly minimize the need for keeping a large number of flow entries in the intermediate forwarding nodes, the ingress/edge nodes still form a bottleneck due to the extra header size required to encapsulate a multi-hop route. OpenFlow defines the MPLS label with 32-bits which make the process of encapsulating each flow packet with the routing path introduces extra transmission overhead, especially in large-scale networks, resulting in bandwidth waste. Source routing also spoils the advantages of per-flow route selection in adaptive to the dynamic QoS demands and load balancing. Moreover, all the existing studies do not provide a clear study on how to traffic engineering SR mechanisms to fit with the proactively installed flow entries under dynamic traffic and topology conditions.

D. ADAPTIVE FLOW ENTRY'S TIMEOUT

In OpenFlow 1.3, the flow entry's idle timeout value (minimum of 1 second) is statically configured to remove entries of inactive flows from the flow tables to offer space for the entries of the most recent flows. The static configuration of the idle timeout can significantly increase the number of flow-rules placement requests to the controller due to the lack of estimating the precise per-flow inter-arrival and transmission intervals. The coarse-grained configuration of the idle timeout is usually inefficient since some flow entries of inactive flow can be still cached. Permanent placement of flow entries can resolve this issue; however, the limitation in flow tables resources cannot accommodate the huge amount of active flow entries without minimizing the flow granularity. On the other hand, OpenFlow 1.4 allows to automatically remove entries of lower importance to offer space for newer flows. However, flow tables can still be easily overflowed and result in a flow entries eviction of some active flows, which in turn, degrades the network performance and scalability. The fixed value of the idle timeout of each flow entry can be adaptively adjusted according to the current network conditions. The timeout can be reactively configured by the controller to give certain priority for adding/removing flow entries, so the flow table can always serve the most significant flows.

In this scheme, maintaining a fine-grained flow control is crucial to classify each flow and decide on how long the flow entry should reside in the flow table. As a result, some research works propose to utilize the logically centralized controller to adaptively configure the flow entry's idle timeout in response to the *current* traffic matrix, see Table 7.

In this essence, the research works in [115], [116] and [117] propose to dynamically adjust the timeout of flow entries. The authors in [115] propose an Adaptive Hard Timeout Method (AHTM) to dynamically adjust the flow entry's hard timeout according to the number of interrupted flows. However, using hard timeout may lead to a large amount of flow interruption in the case of dynamic traffic conditions and can cause a rule removal during the transmission of a burst

TABLE 7. Comparison of the proposed openFlow-SDN adaptive flow entry's timeout mechanisms.

Proposed Approach	Rules Placement	Method	Fine-grained	Objectives	Cons.
AHTM [115]	Reactive	Adaptive Timeout	✓	Dynamically adjusts the flow entry's hard timeout according to the number of interrupted flows (blocking probability) to improve flow table utilization	Using hard timeout can cause a rule removal during the transmission of a burst of packets, which increases latency and damages network performance
TimeoutX [116]	Reactive	Adaptive Timeout	✓	Sets an adaptive timeout to each flow according to its estimated duration, flow type, and the current flow table utilization ratio	Has no eviction mechanism for non-significant and finished flows with valid idle timeout in the case of highly-active networks
Liang et al. [117]	Reactive	Adaptive Timeout	✓	Uses the invalid lifetime of flow entry to represent flow table resource cost and packet_in events generated per seconds to represent the controller processing resource cost to calculate the effective idle_timeout value	Has no eviction mechanism for non-significant and finished flows with valid idle timeout in the case of highly-active networks
TF-IdleTimeout [118]	Reactive	Adaptive Timeout	✓	Depends on some factors in addition to the flow inter-arrival such as; flow rate and flow type to dynamically set the appropriate idle_timeout	Has no eviction mechanism for non-significant and finished flows with valid idle timeout in the case of highly-active networks
Liu et al., [119]	Reactive	Adaptive Timeout	✓	Adjusts the timeout value for the new coming flow according to the current estimated remaining resources in the flow tables	Has no eviction mechanism for non-significant and finished flows with valid idle timeout in the case of highly-active networks
Guo et al., [120]	Reactive	Adaptive Timeout	✓	Detects the real-time flow tables utilization for evicting expired flow entries when needed to accommodate new flows	

of packets, which increases latency and damages network performance. The authors in [116] propose to set an adaptive timeout to each flow according to its estimated duration, flow type, and the current flow table utilization ratio. The authors in [117] focus on optimizing idle_timeout value to utilize network resources in OpenFlow-based networks efficiently. Based on the ON/OFF traffic model, they experiment and analyze the influence of static idle timeout value in both flow tables and the controller. The invalid lifetime of flow entry is used to represent flow table resource cost, where *packet_in* events generated per seconds are used to represent the controller processing resource cost. These two resource cost indicators represent the upper and lower bound of idle timeout which can be used to calculate the effective idle timeout value of flow entries. The research work in [118] also depends on some factors in addition to the flow inter-arrival such as; flow rate and flow type to dynamically set flow entry's idle timeout. The authors in [119] also propose to dynamically adapt the idle_timeout to adjust the timeout value for the new coming flow according to the current estimated remaining resources in the flow tables. The proposed mechanism, first estimates how many flow entries could survive to the next sampling time. Then it estimates the number of new flows that may need to be installed in the next sampling time. As such, the proper timeout value can be estimated to satisfy that the rest of new flows at the next sampling moment are less than the remaining resources of flow tables.

Previously proposed solutions did not provide a mechanism to evict non-significant flow entries in the case of highly-active networks. As a result, the authors in [120] propose to control the idle timeout value of each flow dynamically. It proposes an online routing scheme so-called Software-defined Adaptive Routing (STAR) that can detect the real-time flow tables utilization for evicting expired flow entries when needed to accommodate new flows. STAR uses the LRU replacement algorithm and idle timeout to allow each switch to remove flow entries. Each flow entry is associated with a binary flag to indicate whether the entry is active or inactive. This flag is set to active when the controller places a new flow entry and set to inactive when the switch receives the last packet (FIN packet) of the flow. As such, the controller can track the actual flow tables utilization by estimating the counter of active-flows and in turn, remove the inactive flows even before their expiry timeout.

Although dynamic flow entry's idle timeout can adaptively add/remove flow entries in response to the dynamic network resource constraints, it cannot adaptively set the idle timeout based on the traffic patterns and expected flow transmission rate.

E. PREDICTIVE FLOW RULES PLACEMENT

Identifying and predicting traffic patterns to place, update or remove flow entries proactively is crucial to reduce the overhead of a large number of flow-rules placement requests and guarantee an efficient resource utilization. The main idea is to proactively install the flow entries of the predicted frequency and elephant flows while removing flow entries of rarely and light flows to give a space for more priority flows in the flow tables and reduce the frequent reactive flow-rules placement requests traffic to the controller. Predicting OpenFlow-SDN traffic patterns can also help in obtaining optimal routing calculation to avoid traffic congestion and therefore enhance

TABLE 8. Comparison of the proposed openFlow-SDN predictive flow rules placement mechanisms.

Proposed Approach	Rules Placement	Method	Fine-grained Control	Prediction Method	Objectives	Cons.
Atlas [121]	Proactive	Predictive Flow Rules Placement	✓	Crowd-sourcing	Relies on deployed agents to collect information about active network sockets of each running application and sent them to the controller's classifier for training data with other flow features such as packet size	Employing agents for collecting application-level information can add more complexity and traffic in large-scale networks
vTC [122]	Proactive	Predictive Flow Rules Placement	✓	Pool of ML algorithms	Dynamically choose the appropriate flow features and classifier to maximize accuracy and reduce classification delay	Cannot work properly for real-time flow that needs a fast rule placement decision
Kim et al. [123]	Proactive	Predictive Flow Rules Placement	✓	Auto-Regression (AR)	Predicts the number of new flows and estimates the number of remaining flows to dynamically adjusts the timeout value of each flow to reserve space for the newly arrived flows in advance	Adjusting the idle timeout does not consider the priority of critical flows which may expose to eviction
TSDN [124]	Proactive	Predictive Flow Rules Placement	✓	Transition Tensor model	Introduces a transition tensor model for network traffic prediction and QoS provisioning	Requires huge historical data to predict the traffic flow from one source to a destination
PFIM [125]	Proactive	Predictive Flow Rules Placement	✓	Periodically check the samples frequency	Dynamically learns the periodic patterns of IoT traffic and accordingly install the appropriate flow entries to the flow-based switch before new packets arrivals	Non-regular interval flow patterns require more complex learning-based algorithms to predict the IoT based traffic pattern
FlowSeer [126]	Proactive	Predictive Flow Rules Placement	✓	Streaming mining	proposes an elephant flow detection and scheduling mechanism that monitor and train the first few packets to predict the rate and duration of the initiated flow under dynamic network and traffic conditions	The cooperation prediction requires to upgrade the data plane which contradicts with the main idea of SDN to keep data plane simple for performing the basic networking tasks

network scalability and throughput. The controller has to simultaneously gather per-flow measurements and statistical information to measure the network accurate and timely traffic matrix. The traffic matrix is the key factor in predicting the future traffic trends to place flow entries optimally.

Due to the burst nature of large-scale networks traffic, network-wide learning-based traffic measurement and classification is crucial for predicting network flow patterns to utilize network resources efficiently. OpenFlow protocol has a built-in data collection that can provide some basic flow information such as packet size, timestamps, inter-arrival time, source and destination MAC/IP/Port, flow duration, byte count and packet count. Furthermore, the centralized visibility of the SDN controller provides a new opportunity for monitoring and predicting network performance. Hence, leveraging the global network view knowledge of the OpenFlow-SDN controller to predict the flow patterns and therefore achieve better QoS provisioning and resource utilization has gained increasing interest among researchers, see Table 8.

To adaptively providing desired QoS for different traffic flows in OpenFlow-base SDN, it is also necessary to monitor and classify network flows at the application level. Therefore, when a new flow arrives, the application-layer classifier can get the flow's features and compares them with the learned features, result in an optimal controller forwarding rule decision. Unfortunately, OpenFlow-based networks are only capable of monitoring Layer 2/3/4 header field of packets and still lack in higher layers application awareness. The OpenFlow-SDN controller can only perform forwarding decisions based on the shortest path or load balancing. Data set and classification features in OpenFlow-SDN must be carefully selected. Therefore, it requires Layer 7 fine-grained application awareness to achieve more intelligent forwarding decisions to give more flow-rules placement priority for real-time or large applications flows and guarantee better QoS control and provisioning. Deep Packet Inspect (DPI) based techniques are quite effective due to its high accuracy measurement level [121]. However, DPI consumes a lot of CPU resources and not effective for the encrypted traffic, which makes it unreliable for large-scale and complex networks of heterogeneous and enormous traffic conditions. On the other hand, a machine learning-based approach does not require packet payload inspection, and only some flow features are selected carefully for training the classifier. The ML-based approach can perform classification in a much lower computational cost than DPI approach, but in a lower accuracy rate due to its coarse-grained classification. Nevertheless, the ML-based technique can be more effective in OpenFlow-SDN than traditional networks due to the global network visibility of the SDN controller and the collected statistical information of each flow entry.

Assume that no application signature is available, the accuracy problem of ML-based and fine-grained application-level classification in OpenFlow-SDN has been addressed in some research works such as [121] and [122]. Research work as

Atlas [121] proposes to use C5.0 decision tree ML-based technique and employ a crowd-sourcing labeling approach on OpenFlow-SDN controller to obtain a fine-grained and accurate application classifier. Atlas relies on deployed agents in some employee devices to collect information about active network sockets of each running application. These formations are sent to the controller where the ML-based classifier is running to correlate and compose the ground truth training data with other flow features such as packet size that are collected by the OpenFlow protocol. Therefore, when new guest devices join the network, OpenFlow sends the flow features to the classifier which can detect the application and recommend the appropriate actions to be installed by the controller. The work gives a solution for an accurate and reliable application awareness ML/OpenFlow based SDN classifier that get 90% accuracy of only a test of 40 applications in Google Play. However, the idea of employing agents for collecting application-level information can add more complexity and traffic in large-scale networks and may not be an optimal solution in IoT network where data source devices are of resource constraints and heterogeneous platforms.

The Accuracy of an ML-based network traffic classifier can also be affected by the type of selected flow features under a highly fluctuating traffic over time/space. Hence, the network flow measurement has to be agile enough to estimate both large and small traffic flows dynamically. SDN has opened up new opportunities to collect per-flow fine-grained measurements of specific features dynamically on-the-fly and to cope with the dynamic network and traffic conditions. According to [122], selecting some flow features for certain ML-based classification techniques can be more effective in achieving higher classification accuracy than other classification techniques. As a result, [122] proposes a software-defined traffic classification framework (vTC) that dynamically choose the appropriate flow features and classifier to maximize accuracy and reduce classification delay. The author uses a pool of ML-based classification models such as K Nearest Neighbors, Support Vector Machine (SVM), Decision Tree (DT), Adaptive Boosting, Naive Bayes and Multi-Layer Perception (MLP) to test the accuracy of each model in categorizing network flows. The results show that the accuracy of the classifier is highly dependent upon the selected flow features and protocol.

Some other research works concentrate on predicting dynamic OpenFlow-SDN network traffic patterns to enable adaptive QoS provisioning and enhance the network's resources utilization such as [123], [124]and [127]. To avoid the eviction of unfinished flows, the authors in [123] propose an algorithm to determine the number of flow entries that are likely to be remaining in the flow table at the next sampling period. As such, it dynamically adjusts the timeout value of each flow to reserve space for the newly arrived flows in advance. The proposed algorithm uses AutoRegression (AR) to predict the number of new flows and utilize the Weibull distribution to estimate the number of remaining flows. Research work as [124] uses the mathematical tensor model that widely used in big data application to proposes a tensor-based SDN (TSDN) model for efficient QoS provisioning in OpenFlow-SDN. The High-order singular value decomposition (HO-SVD) method and incremental updating approach are employed to extract the most valuable flow entry header fields and generate the forwarding tensor. The dynamic update in the network topology can be rapidly reported to the controller by incrementally updating the forwarding tensor. The incremental tensor decomposition approach is employed to generate the core tensor which contains the most valuable forwarding information. All the core tensors generated in the data plane are submitted to the control plane to generate the controlling tensor which capable of globally computing optimal paths for data packet scheduling. The transition tensor model which consists of four orders and uses the eigenvalue decomposition method to compute the stationary distribution is also proposed to predict the network traffic.

In OpenFlow-SDN, the controller needs to calculate and install the forwarding rule of each unknown stream in real-time, which may cause a computation and communication burden on both data and control planes in large-scale networks [128]. In this essence, the authors in [125] propose a Pre-Emptive Flow Installation Mechanisms (PFIM) to reduce the load on the controller result from the burden flow requests traffic. PFIM dynamically learns the periodic patterns of IoT traffic and accordingly install the appropriate flow entries to the flow-based switch before new packets arrivals. This work only focuses on monitoring the packets arriving time of the new flows at the forwarding device to identify flows of regular intervals. However, non-regular interval flow patterns require more complex learning-based algorithms to predict the IoT based traffic pattern.

Other research works such as FlowSeer [126], and [129] adopt learning algorithms to predict the dynamic traffic patterns of OpenFlow-SDN large scale networks. FlowSeer [126] proposes an elephant flow detection and scheduling mechanism that monitor and train the first few packets to predict the rate and duration of the initiated flow under dynamic network and traffic conditions. However, FlowSeer does not clearly explain the flow features selected in the learning data set as inputs. Furthermore, the proposed a cooperative prediction mechanism that enables the switch to perform most of the classification decisions, requires to update the data plane which contradicts with the main idea of SDN to keep data plane abstract and simple for performing the basic networking tasks. Elephant flow prediction mechanism is also proposed in [129] to meet with the demands of the traffic characteristics in data center networks. Both research studies focus on how to reduce the controller to switches communication overhead by predicting the elephant flows and adapt their routing policies to meet with the dynamic network conditions demands. However, none of them study the scalability of the network when connecting to more complex and dynamic large-scale networks as IoT, large virtualized data centers and multi-tenant cloud networks.

Taking into consideration the network-wide state knowledge of the SDN controller is important for achieving an optimal traffic measurement and performance prediction. Hence, more research is required in selecting more valuable flow features as a training set for predicting the flow patterns. Thus, proactively set up the flow to minimize the control traffic overhead on the controller that caused due to the huge amount of flow-rules placement requests in the case of dynamic and large-scale networks.

VII. OPENFLOW-SDN CONTROLLERS LOAD BALANCING

In the OpenFlow-based distributed SDN control plane architecture, the mapping between the data forwarding nodes and controllers is statically configured. Any workload changes in the data plane can easily lead to load imbalance in the distributed controllers. The controller can be overwhelmed by the high amount of flow-rules placement requests and inefficient resource utilization due to uneven load balancing among controllers under spatial and temporal variations in the traffic conditions. Hence, the assignments of the data forwarding nodes to the controllers need to be carefully configured to prevent controllers from being overloaded. In OpenFlow protocol, the problem of the overloaded controller is partially resolved by assigning two controllers with different roles (master, equal or slave) for each switch. If the master controller fails due to overload or other exceptions, the role of the equal controller can be changed to master. However, this mechanism is not adaptive and robust enough to the controller's failure, and cannot achieve optimal load balancing and resource utilization in the control plane. To resolve this issue, the authors in [130] propose a dynamic slave controller assignment that prevents the network crash by planning slave controllers assignment ahead of the controller failures.

The hierarchically distributed controllers has been also proposed to partition the load and functionalities among controllers as in Kandoo [14], Xbar [15], ORION [16], [48], [131] and [132]. Although the hierarchically distributed controllers can improve the network scalability and partition the traffic load and functionality into level-based controllers, it cannot optimally distribute the load among the lower-level controllers of local network state view. The local area controllers are statically mapped to their local-view data plane forwarding nodes and do not consider the current state and load fluctuation of the network in its static load partition. As such, both the area and domain controllers can still be overwhelmed by the high imbalance aggregated flow in their mapping data plane forwarding nodes.

Elastic controller provisioning in response to the temporal/spatial variation in network traffic conditions can also be used to improve scalability and prevent the controller from being a potential bottleneck. However, dynamic resource provisioning without efficiently use the available resources are wasting of resources and improperly increase the capital expenditures (CAPEX). Therefore, proposing an efficient and adaptive load balancing scheme for the control plane gains more interest among researchers.

Wang *et al.* [133] provided a survey on the recent solutions for balancing traffic across links in data center networks. Part of the literature presents the load balancing mechanisms that leverage SDN global link and traffic information to balance the load in the data plane. However, in this survey, we aim at presenting the scalability concern in the control plane due to load imbalance in the logically-centralized controllers. This paper presents the state-of-the-art load balancing mechanism in the control plane that dynamically reassigns controllers to data forwarding planes or flow-rules placement requests to different underloaded controllers. Another survey in [134] presented the proposed techniques toward balancing the load in both the data and control planes. However, we cover also the most recent research works toward the controllers' load balancing and compare the proposed approaches. Finally, we classify the OpenFlow-SDN load balancing in the control plane into two main schemes; dynamic assignment of controllers and dynamic assignment of flow requests, see Figure 13. A comparison of the different controllers' load balancing mechanisms is also provided in Table 9.

A. DYNAMIC ASSIGNMENT OF CONTROLLERS

One of the proposed mechanism for balancing the load among the distributed controllers is to dynamically migrate the data plane forwarding nodes from the overloaded controllers to the underloaded ones. For instance, ElastiCon [135] proposes a switch migration protocol that monitors the load on all controllers and adaptively reassign data plane forwarding nodes to the underutilized controllers. ElastiCon also proposes elastic provisioning of the controllers from a pool of controllers that dynamically grow or shrink in response to the aggregated load capacity threshold of all existing controllers. To guarantee liveness, ElastiCon lets the controller remain active when migration happens until the switch finishes the command processing. ElastiCon implements a new 4-phase migration protocol to make sure that at least one controller is active (master or equal mode) and hence minimize disruption to ongoing flows, and automatically fit with the spatial and temporal variations in the flow conditions.

Partitioning or distributing network state views among controller instances is used to avoid overwhelming the controller resources due to replicating network state at all controller instances. However, it can unacceptably increase flow-rules placement propagation latency as a result of inter-controller communications to access the topology state. An experiment commenced by Pratyaastha [136] to measure the RTT of the first packet of each new flow shows that computation time requires for flow-rules placement is negligible while the key factor in the high flow-rules placement latency is the time to access remote state from the distributed network-wide state. To address this problem, Pratyaastha proposes an approach for assigning switches to controllers and partition the network state views among the distributed controllers by minimizing

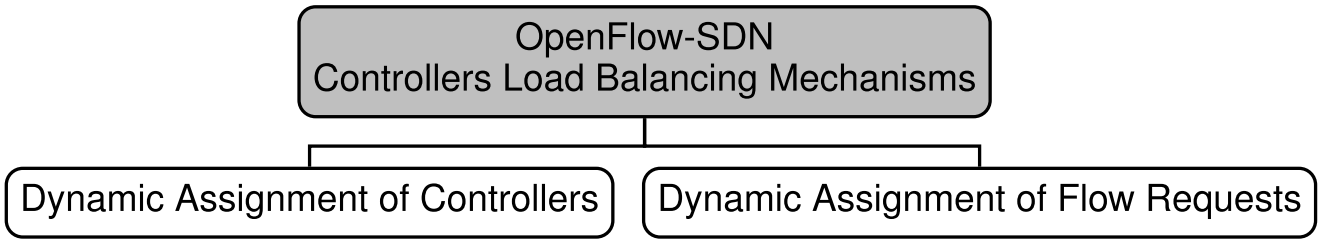

FIGURE 13. OpenFlow-SDN controllers load balancing mechanisms.

TABLE 9. Comparison of the proposed openFlow-SDN controllers load balancing mechanisms.

Proposed Approach	Method	Low Reconfiguration Overhead	Efficient Resource Utilization	Objectives	Cons.
ElastiCon [135]	Dynamic Assignment of Controllers			Dynamically re-assign data plane forwarding nodes to the underutilized controllers	Did not describe how to select a migrated switch and its target controller
Pratyaastha [136]	Dynamic Assignment of Controllers			Assigns switches to controllers and partition the network state views among the distributed controllers by minimizing the number of dependencies between SDN switches and a specific state partition	Introduces a reconfiguration overhead due to a sharp and frequent controller to switch reassignments
Cheng et al. [55]	Dynamic Assignment of Controllers		✓	Controllers with residual resources compete to fully utilize its resources by achieving switches migration, while overloaded controllers migrate their tasks to avoid service interruption	Measuring the resource utilization by the controllers without considering the relation between switches and controllers is not enough to achieve efficient dynamic controllers assignment
Ye et al. [56]	Dynamic Assignment of Controllers		✓	Proposes a multi-dimensional resource-consuming model that considers switches as resources consumers and controllers as resource providers	In highly dynamic network, it can suffer from the frequent threshold-based switch migration
Wang et al. [137]	Dynamic Assignment of Controllers	✓		Minimizing the migration cost by calculating the migration cost and efficiency to balance the trade-off between the migration cost and the load balance rate	Suffers from large execution time and cannot guarantee efficient resource utilization
Wang et al. [57]	Dynamic Assignment of Controllers	✓		Minimizing the total cost caused by the response time and controllers maintenance by re-assigning the forwarding nodes in a timely fashion in response to the dynamic variation of network conditions	Suffers from large execution time and cannot guarantee efficient resource utilization
Hu et al. [138]	Dynamic Assignment of Controllers	✓		Migrates switches according to the selection probability, and determines the target controllers by calculating the migration cost of data collection, switch migration and controller state synchronization	Suffers from large execution time and cannot guarantee efficient resource utilization
ASIC [139]	Dynamic Assignment of Flow Requests	✓		Uses a load balancer to select the appropriate intra-domain controllers for processing flow-rules requests in parallel	The use of proxy load balancer can form a potential single point of failure
BalanceFlow [140]	Dynamic Assignment of Flow Requests	✓		Collecting flow requests information from all controllers by a super-controller to generate the re-assignment rules and install them to the flow tables of the corresponding forwarding nodes	Adds more configuration and monitoring traffic overhead
Cimorelli et al. [141]	Dynamic Assignment of Flow Requests	✓		Makes switches to estimate the controllers' state according to the delay function and select the available controller to process the flow-rules placement requests	Need to upgrade the forwarding element to support the proposed solution

the number of dependencies between SDN switches and a specific state partition. However, in highly dynamic workload conditions, resource requirements and state partitions can dynamically change and hence introduces a reconfiguration overhead due to a sharp and frequent controller to switch reassignments. Moreover, such deterministic approach that depends on a specific threshold needs is inflexible to some extent for variable network environment. Therefore,

an adaptive reassignment mechanism is required to reduce the frequent disruption in SDN network configuration.

The research works in [55]–[57], [137] propose the online migration of switches to controllers to utilize available resources and balance a load of controllers efficiently. Some research works [55], [56] focus on how to efficiently utilize resources (CPU, bandwidth, and memory) to achieve load balancing among controllers. Cheng *et al.* [55] as an instance, proposes a GAME-Switch Migration (GAME-SM) mechanism to carefully migrate a small number of switches among controllers for each time of adjustment to efficiently maximize resource utilization with the dimension of CPU, bandwidth, and memory. Unlike ElastiCon, the authors in [55] propose to let both controllers with residual resources and the overloaded controller to play a role in the load balancing and resource utilization. Inspiring by the power of game theory, the controllers with residual resources will act as players that compete to fully utilize its resources by achieving switches migration. On the other hand, overloaded controllers will try to migrate their tasks to avoid service interruption. Measuring the resource utilization by the controllers without considering the relation between switches and controllers is not enough to achieve efficient dynamic controllers assignment. As a result, Ye *et al.* [56] propose a multi-dimensional resource-consuming model that considers switches as resources consumers and controllers as resource providers. Since different data plane forwarding events have uneven resource demands such as CPU, bandwidth and memory, the migration decision is achieved based on how the model can obtain load balancing among these multi-dimensional resources to maximize resource utilization. For instance, a controller that control more edge forwarding nodes needs more CPU resources than others due to the high number of flow request for unknown new flows. Core forwarding nodes, on the other hand, consume more bandwidth resources than others due to the high amount of aggregated traffic.

The dynamic online assignment of controllers has to consider the trade-off between the load balance rate and migration cost and elaborately decide which forwarding node and where it should be migrated to avoid service interruption. Some research works as in [57], [137], [138] study the dynamic controller assignment problem (DCAP) and proposes solutions to minimize the migration cost in production networks as in data centers. For instance, the research work in [137] proposes an efficient-aware switch migration-based decision-making (SMDM) algorithm which based on the greedy method. SMDM performs the migration process in three main phases. First, it monitors the aggregated load in each controller and measures the load diversity on the controllers to decide on whether to perform the migration. Second, it calculates the migration cost and migration efficiency to prepare possible migration choices. The migration efficiency of moving a switch to a controller is defined as a ratio of load balance variation to the migration cost of the increase in load cost and message exchange cost. Finally, it prepares a migration plane for migrating some switches to controllers with higher migration efficiency.

Another study in [57] formulates the dynamic controller assignment problem (DCAP) as an online optimization to minimize the total cost caused by the response time and controllers maintenance. To let the forwarding nodes to be re-assigned in a timely fashion in response to the dynamic variation of network conditions, it proposes a two-phase offline algorithm that uses the concepts of both matching theory and conditional games. In the first phase, forwarding nodes are defining their preferences over controllers based on the worst response time that the controller can provide, and in turn, the controllers are defining their preferences based on the control traffic overhead caused by the communication between them. In the second phase, the outcomes from phase one are used to improve the matching solution by using a coalitional game approach that further reduces the response time. Then applying both the two-phase offline algorithm and the Randomized Fixed Horizon Control (RFHC) framework to develop an online algorithm for efficiently solving the switch-controllers assignment problem. Distributed Decision Mechanism (DDM) [138] is another research work that migrates switches according to the selection probability, and the target controllers are determined by calculating the migration cost of three integrated cost factors; data collection, switch migration and controller state synchronization.

Although the dynamic assignment of switches to controllers can improve the scalability of SDN, in very active networks, the determination of the exact moment of controllers' hand-off in a disruption-free manner between controllers can be minimal and useless. As a result, the dynamic re-assignment of the forwarding nodes to controllers disturb the ongoing flows and introduce liveness, consistency and reliability concerns.

B. DYNAMIC ASSIGNMENT OF FLOW REQUESTS

Another attempt toward controllers load balancing is to dynamically redirect the flow-rules placement requests from the overloaded to underloaded controllers while preserving the static mapping between control and data planes.

ASIC [139] proposes to redirect the flow-rules placement requests to a load balancer to select the appropriate intra-domain controllers for processing these requests in parallel. However, the use of proxy load balancer can form a potential single point of failure. BalanceFlow [140] also proposes a controller load balancing architecture for the OpenFlow-based wide-area networks that can dynamically partition flow requests among controllers. Every controller in BalanceFlow shares the same network-wide replica and publishes its load information periodically through a cross-controller communication system. Unlike ASIC, a super controller in BalanceFlow is only responsible for collecting the periodically published flow requests information from all controllers. Next, it partitions the traffic in response to the collected information and generates the assignment information. Finally, the assignments rules are

installed to the flow tables of the corresponding forwarding nodes. To assign each new flow-rules placement request to a certain controller, BalanceFlow extends OpenFlow by introducing the CONTROLLER *X* action to forward the new flow to the under-loaded *X* controller. It installs all the potential flows and their wildcard matching ingress-egress route pairs with a permanent idle-timeout to the forwarding nodes. As such, the first packets of new flows always match allocation rules and are sent to different controllers to achieve controller load balancing. BalanceFlow also presents a traffic partition heuristic to avoid propagation latency of allocating flow-requests to distant remote controllers. The drawbacks of BalanceFlow is that it adds more configuration and monitoring traffic overhead and a single super-controller for load balancing may not perform well in large-scale networks.

The authors in [141] also propose a distributed load balancing algorithm for the control traffic based on the game theory and converges to a specific equilibrium known as Wardrop equilibrium. The switches estimate the controllers' state according to the delay function and select the available controller to process the flow-rules placement requests.

Controller elasticity and load balancing suffer from two main problems in the case of variant traffic conditions which can hinder scalability and performance of SDN. First, the reactively threshold-based provisioning of new controllers can introduce an overhead due to sharp and frequent controller instances provisioning and shrinking. This is because of the enormous traffic that can be generated by control and data planes to maintain load-balancing, consistency, and global domain network state among controllers. Second, partitioning network state views among controllers introduce a flow-rules placement delay due to inter-controller communication to access applications or topology state. This overhead can only be acceptable if forwarding rules are proactively installed, however it can still disrupt many critical applications that depend on fast reactive forwarding rule installation such as consistent load balancing, traffic engineering, and dynamic traffic filtering. More researches are required to be invested in proactive and soft controller provisioning and flow-rules placement distribution, that can minimize controller consistency overhead and balance load among controllers.

VIII. RESEARCH CHALLENGES AND FUTURE DIRECTIONS

Decoupling control logic from data forwarding nodes to enable centralized management introduces new scalability and performance challenges in the SDN. Very limited research works have discussed the performance and scalability of the OpenFlow-SDN in dynamic and large-scale networks like the Internet of Things (IoT) where a huge number of entities (e.g., physical objects, VMs, applications) communicate and dynamically join or leave the network. In this section, we present some of the scalability and performance challenges that encounter the OpenFlow-SDN and deserve more research efforts.

A. CHALLENGES RELATED TO OPENFLOW-SDN CENTRALIZED VISIBILITY

To obtain centralized and optimal network flow management and configuration, controllers have to maintain a global view of the network topology graph. Replicating the global link-state view in every distributed controller can guarantee a logically-centralized controlling and transparency over the data plane. Moreover, it can prevent the routing misbehavior caused by controllers' inconsistency in the interval between two consecutive synchronizations. However, maintaining a consistent replica of the network-wide state among controllers in a large-scale network of dynamic traffic and topology conditions can result in a massive amount of frequent synchronization traffic that can overwhelm the controller.

Previous literature proposed solutions that addressed different consistency and synchronization mechanisms to improve SDN scalability. Eventual consistency can guarantee faster controller response time but it can lead to potential routing problems (e.g., routing loops, black holes). On the other hand, strong consistency can avoid routing problems but with the cost of higher delay, lower availability and computational complexity overhead. Adapting consistency in according to the current network state can maintain scalability that sacrifices application optimality for less synchronization overhead. However, in highly dynamic networks, it requires more computational complexity to monitor and measure the traffic metrics for obtaining adaptive controllers' consistency. Event-driven synchronization can also reduce the consistency traffic by synchronizing only the updated controller state according to the data plane topology change and link failure events.

Partitioning the controllers into clusters and using eventual consistency in response to data plane changes events is a practical and agile solution in the large-scale and dynamic SDN networks. However, the packets that are traveling on the failed link during inconsistency interval will be lost and dropped until the assigned controller recalculate and install the alternate forwarding rules to recover the network failure. Therefore, it is crucial to enable OpenFlow-SDN to bypass the link failures in less recovery time by providing alternate routing paths in advance to avoid delay and communication traffic overhead imposed by strong consistency.

B. CHALLENGES RELATED TO OPENFLOW-SDN DISCOVERY PROTOCOL

Previous literature does not give a robust study and analysis on the effect of discovery protocol in the context of large-scale SDN-based networks of dynamic traffic and topology conditions. Furthermore, the proposed solutions toward hybrid-SDN discovery either use the existing LLDP protocol or propose a new protocol. However, none of these studies proposed a solution to enable OpenFlow to discover non-OpenFlow elements in hybrid-SDN, so the controller can get statistical information about adjacent non-OpenFlow topology for achieving optimal routing decisions on the

border forwarding elements. Therefore, there is a need for more research efforts to propose an efficient OpenFlow-based discovery protocol in the context of hybrid-SDN. More research efforts also required to enhance the discovery process to be triggered asynchronously in response to reactive or predicted topology changes to save network resources and avoid sending unnecessary redundant topology information to the controller.

In addition, the control plane has to have a real-time up-to-date view of the global network view and status to act as a centralized controller and efficiently serve the data plane forwarding requests. The entire control plane management procedure is substantially affected by how efficiently it can discover data plane forwarding nodes and links to maintain a centralized global view of the network topology. Therefore, more research efforts are necessary to study the performance of the OpenFlow discovery under large-scale and dynamic network conditions.

C. CHALLENGES RELATED TO OPENFLOW-SDN FLOW-RULES PLACEMENT

In a highly dynamic and large-scale network, the number of flow-rules placement requests to the controller increase rapidly, which can overwhelm the controller and delay the forwarding process. This delay can significantly increase to a level which can not meet with the requirements of real-time applications and result in degrading network performance and scalability. The OpenFlow controller can proactively populate the flow entries of all traffic matches in advance to the flow arrival. However, in large-scale networks, the TCAM-based flow tables can be overflowed by the high number of installed flow entries that match the traffic between all ingress and egress ports. Previous literature proposed solutions such as efficiently using memory resources in the data plane, bring back some control logic to the data plane, use per-flow source routing or adaptively adjust the flow entries idle timeout. However, very few research efforts focus on predicting the flow patterns to install the forwarding rules in advance to the arrival of flows. Most of the proposed learning-based mechanisms focus only on monitoring and classifying the SDN traffic. Therefore, more research is required in selecting more valuable flow features as a training set for predicting the flow patterns. Thus, controllers can proactively install the forwarding rules to minimize the communication traffic overhead between the control and data planes.

The proposed solutions toward dynamically adjusting the flow entries idle timeout focus on measuring the current network traffic performance without considering the application and QoS demands. There are also a lot of dynamic applications whose traffic transmission inter-arrival cannot be accurately estimated or predicted. For example, in on-demand applications (e.g., VoIP and video-on-demand), flows come and go unexpectedly, and their bandwidth requirements and duration cannot be known in advance. Hence, it is also necessary to monitor and classify network flows at the application level to dynamically adjust flow entries' timeout.

D. CHALLENGES RELATED TO OPENFLOW-SDN CONTROLLERS LOAD BALANCING

Under spatial and temporal variations in the traffic conditions, the controller can be overwhelmed by the high amount of flow-rules placement requests and inefficient resource utilization due to uneven load balancing among the logically centralized controllers. Most of the proposed solutions toward controllers load balancing focus on either dynamically reassigns controllers to the forwarding elements or dynamically reassign flow-rules placement requests to the underloaded controllers according to the dynamic variation of network conditions. A sharp and frequent migration of forwarding nodes to controllers can introduce a reconfiguration overhead which can be avoided by calculating the migration cost and decide accordingly. However, most of the proposed solutions suffer from large execution time and cannot guarantee efficient resource utilization. A better solution to the reconfiguration overhead due to reactive migration is to predict the load imbalance and act accordingly. On the other hand, most of the proposed solutions toward reactively reassigning flow-rules request to the underloaded controllers suffer from high response time due to propagation delay and computational complexity. Moreover, none of the proposed solutions study the possibility of proactively reassigning flow entries based on the predicted load imbalance in controllers.

IX. CONCLUSION

Software-Defined Networking (SDN) is an emerging network architecture that promises to simplify network management, improve network resource utilization, and boost evolution and innovation in traditional networks. SDN allows the abstraction and centralized management of the lower-level network functionalities by decoupling the network logic from the data forwarding devices into a logically centralized distributed controllers. However, in highly dynamic large-scale networks, this separation introduces two types of communication overhead: 1) control traffic overhead between control and data planes, and 2) consistency traffic overhead between distributed controllers to maintain logically-centralized control over the network. This traffic can lead to problems such as controller overloading, inefficient resource utilization, routing problems, high controllers response time and data plane memory overloading. As such, impact the overall performance and scalability of SDN.

In this survey, we have presented four main challenges in OpenFlow-SDN that can be considered as primary sources of such traffic overhead. The survey, have first given an overview of OpenFlow-SDN flow control and presented some critical challenges in the OpenFlow-SDN flow control that affects the performance and scalability of the network such as logically-centralized visibility, link-state discovery, flow-rules placement problem and controllers load balancing. We have then discussed each issue and presented the related existing solutions and limitations in enhancing the OpenFlow-SDN scalability and performance. Finally,

we have outlined the research challenges that need to be addressed further toward more adaptive and scalable OpenFlow-SDN solutions under large-scale and dynamic network conditions. These challenges include the ability of OpenFlow-SDN to bypass link failures in less recovery time, discover non-OpenFlow elements in the context of hybrid-SDN, avoid sending redundant topology information to the controllers, monitor and classify network flows at the application level to dynamically adjust flow entries' idle timeout, and proactively assign flow entries based on the predicted load imbalance in controllers.

ACKNOWLEDGMENT

The authors would like to thank the Universiti Teknologi Malaysia for providing the research environment and facilities to support accomplishing this research. The first author would like to thank Dr. Mohd Murtadha for supporting him during his Ph.D.

REFERENCES

[1] D. Kreutz, F. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[2] W. Braun and M. Menth, "Software-defined networking using OpenFlow: Protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014. [Online]. Available: http://www.mdpi.com/1999-5903/6/2/302

[3] A. Al-Shabibi, M. D. Leenheer, M. Gerola, A. Koshibe, W. Snow, and G. Parulkar, "OpenVirteX: A network hypervisor," in *Proc. Open Netw. Summit (ONS)*, 2014. [Online]. Available: https://www.usenix.org/conference/ons2014/technical-sessions/presentation/al-shabibi

[4] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Comput. Netw. J.*, vol. 72, pp. 74–98, Oct. 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614002588

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013. doi: 10.1145/2534169.2486019.

[6] S. Singh and R. K. Jha, "A survey on software defined networking: Architecture for next generation network," *J. Netw. Syst. Manage.*, vol. 25, no. 2, pp. 321–374, 2017. doi: 10.1007/s10922-016-9393-9.

[7] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 136–141, Feb. 2013.

[8] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: From concept to implementation," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 2181–2206, 4th Quart., 2014.

[9] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Comput. Netw.*, vol. 112, pp. 279–293, Jan. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138912861630411X

[10] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proc. 9th USENIX Conf. Operating Syst. Design Implement.*, 2010, pp. 351–364. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924968

[11] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain SDN controllers," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–4.

[12] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an open, distributed SDN OS," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2014, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2620728.2620744

[13] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Jun. 2014, pp. 1–6.

[14] S. H. Yeganeh, Y. Ganjali, S. H. Yeganeh, and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. HotSDN 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 19–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342446

[15] J. McCauley, A. Panda, and M. Casado, "Extending SDN to large-scale networks," in *Proc. ONS*, Mar. 2013, pp. 1–2.

[16] Y. Fu, J. Bi, Z. Chen, K. Gao, B. Zhang, G. Chen, and J. Wu, "A hybrid hierarchical control plane for flow-based large-scale software-defined networks," *IEEE Trans. Netw. Service Manage.*, vol. 12, no. 2, pp. 117–131, Jun. 2015.

[17] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in SDN-based core networks," in *Proc. IEEE ITC*, Sep. 2013, pp. 1–9.

[18] S. Lange, S. Gebert, J. Spoerhase, P. Rygielski, T. Zinner, S. Kounev, and P. Tran-Gia, "Specialized heuristics for the controller placement problem in large scale SDN networks," in *Proc. 27th Int. Teletraffic Congr. (ITCs)*, 2015, vol. 12, no. 1, pp. 210–218.

[19] R. L. Gomes, L. F. Bittencourt, E. R. M. Madeira, E. Cerqueira, and M. Gerla, "Bandwidth-aware allocation of resilient virtual software defined networks," *Comput. Netw.*, vol. 100, pp. 179–194, May 2016. doi: 10.1016/j.comnet.2016.02.024.

[20] P. Xiao, Z.-Y. Li, S. Guo, H. Qi, W.-Y. Qu, and H.-S. Yu, "A K self-adaptive SDN controller placement for wide area networks," *Frontiers Inf. Technol. Electron. Eng.*, vol. 17, no. 7, pp. 620–633, 2016. [Online]. Available: http://link.springer.com/10.1631/FITEE.1500350

[21] S. H. Yeganeh and Y. Ganjali, "Beehive: Simple distributed programming in software-defined networks," in *Proc. Symp. SDN Res. (SOSR)*, 2016, pp. 1–12. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2890955.2890958. doi: 10.1145/2890955.2890958.

[22] K. S. Sahoo, D. Puthal, M. S. Obaidat, A. Sarkar, S. K. Mishra, and B. Sahoo, "On the placement of controllers in software-defined-WAN using meta-heuristic approach," *J. Syst. Softw.*, vol. 145, pp. 180–194, Nov. 2018. doi: 10.1016/j.jss.2018.05.032.

[23] H. Lu, N. Arora, H. Zhang, C. Lumezanu, J. Rhee, and G. Jiang, "HybNET: Network manager for a hybrid network infrastructure," in *Proc. Ind. Track 13th ACM/IFIP/USENIX Int. Middleware Conf.*, 2013, pp. 6:1–6:6. doi: 10.1145/2541596.2541602.

[24] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu, "Traffic engineering in SDN/OSPF hybrid network," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, Oct. 2014, pp. 563–568.

[25] Y. Guo, Z. Wang, X. Yin, X. Shi, J. Wu, and H. Zhang, "Incremental deployment for traffic engineering in hybrid SDN network," in *Proc. IEEE 34th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Dec. 2015, pp. 1–8.

[26] Y. Hu, W. Wang, X. Gong, X. Que, Y. Ma, and S. Cheng, "Maximizing network utilization in hybrid software-defined networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec 2015, pp. 1–6.

[27] X. Jia, Y. Jiang, and Z. Guo, "Incremental switch deployment for hybrid software-defined networks," in *Proc. Conf. Local Comput. Netw. (LCN)*, 2016, pp. 571–574.

[28] S. Salsano, P. L. Ventre, F. Lombardo, G. Siracusano, M. Gerola, E. Salvadori, M. Santuari, M. Campanella, and K. Prete, "Hybrid IP/SDN networking: Open implementation and experiment management tools," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 1, pp. 138–153, Mar. 2016.

[29] C. Ren, S. Wang, J. Ren, X. Wang, T. Song, and D. Zhang, "Enhancing traffic engineering performance and flow manageability in hybrid SDN," in *Proc. IEEE GLOBECOM*, Dec. 2016, pp. 1–7.

[30] S. Huang, J. Zhao, and X. Wang, "HybridFlow: A lightweight control plane for hybrid SDN in enterprise networks," in *Proc. IEEE/ACM 24th Int. Symp. Qual. Service (IWQoS)*, Jun. 2016, pp. 1–2.

[31] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu, "Traffic engineering in hybrid SDN networks with multiple traffic matrices," *Comput. Netw.*, vol. 126, pp. 187–199, Oct. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138912861730289X

[32] Z. Guo, W. Chen, Y.-F. Liu, Y. Xu, and Z.-L. Zhang, "Joint switch upgrade and controller deployment in hybrid software-defined networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1012–1028, May 2019.

[33] *SDN Architecture Overview*, Open Netw. Found., Silicon Valley, CA, USA, 2013, no. 1, pp. 1–5.

[34] *OpenFlow Switch Specification 1.5.0*, Standard ONF TS-020, 2013, pp. 1–205.

[35] N. McKeownm, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008. doi: 10.1145/1355734.1355746.

[36] J. H. Doria, J. H. Salim, R. Haas, H. M. Khosravi, W. Wang, L. Dong, R. Gopal, and J. M. Halpern, "Forwarding and control element separation (ForCES) protocol specification," *Sante Publique*, vol. 28, no. 3, pp. 391–397, 2016. [Online]. Available: http://medcontent.metapress.com/index/A65RM03P4874243N.pdf

[37] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in software defined networks," *Comput. Netw.*, vol. 71, pp. 1–30, Jun. 2014.

[38] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008. doi: 10.1145/1384609.1384625.

[39] D. Erickson, "The beacon OpenFlow controller," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 13–18. [Online]. Available: http://doi.acm.org/10.1145/2491185.249118

[40] (2013). *FloodLight Controller*. [Online]. Available: https://floodlight.atlassian.net/wiki

[41] Z. C. Cai, A. L. Ng, and T. S. Eugene. (2011). *Maestro: A System for Scalable OpenFlow Control*. [Online]. Available: https://www.cs.rice.edu/~eugeneng/papers/TR10-11.pdf

[42] A. Voellmy and J. Wang, "Scalable software defined network controllers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, p. 289, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2377677.2377735

[43] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," presented at the 2nd USENIX Workshop Hot Topics Manage. Internet, Cloud, Enterprise Netw. Services, 2012.

[44] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw.*, 2010, p. 3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863133.186313

[45] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: A software defined Internet exchange," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 551–562, Oct. 2014. doi: 10.1145/2740070.2626300.

[46] RYU SDN Controller. *RYU Controller*. Accessed: Jul. 12, 2019. [Online]. Available: https://osrg.github.io/ryu/

[47] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller fault-tolerance in software-defined networking," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, 2015, pp. 4:1–4:12. doi: 10.1145/2774993.2774996.

[48] Y. Fu, J. Bi, K. Gao, Z. Chen, J. Wu, and B. Hao, "Orion: A hybrid hierarchical control plane of software-defined networking for large-scale networks," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, Oct. 2014, pp. 569–576.

[49] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in OpenFlow-based software defined networks," *Comput. Commun.*, vol. 77, pp. 52–61, Mar. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366415003527

[50] G. Tarnaras, F. Athanasiou, and S. Denazis, "Efficient topology discovery algorithm for software-defined networks," *IET Netw.*, vol. 6, no. 6, pp. 157–161, Nov. 2017. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/iet-net.2017.0066

[51] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "Research challenges for traffic engineering in software defined networks," *IEEE Netw.*, vol. 30, no. 3, pp. 52–58, May/Jun. 2016. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7474344

[52] S. Tomovic, K. Yoshigoe, I. Maljevic, and I. Radusinovic, "Software-defined fog network architecture for IoT," *Springer Wireless Pers. Commun.*, vol. 92, no. 1, pp. 181–196, 2017. [Online]. Available: https://doi.org/10.1007/s11277-016-3845-0

[53] S. K. Tayyaba, M. A. Shan, N. S. A. Khan, Y. Asim, W. Naeem, and M. Kamran, "Software-defined networks (SDNs) and Internet of Things (IoTs): A qualitative prediction for 2020," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 11, pp. 385–404, 2016.

[54] C. Metter, M. Seufert, F. Wamser, T. Zinner, and P. Tran-Gia, "Analytical model for sdn signaling traffic and flow table occupancy and its application for various types of traffic," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 3, pp. 603–615, Sep. 2017.

[55] G. Cheng, H. Chen, H. Hu, and J. Lan, "Dynamic switch migration towards a scalable SDN control plane," *Int. J. Commun. Syst.*, vol. 23, no. 5, pp. 633–652, 2016.

[56] X. Ye, G. Cheng, and X. Luo, "Maximizing SDN control resource utilization via switch migration," *Comput. Net.*, vol. 126, pp. 69–80, Oct. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128617302669

[57] T. Wang, F. Liu, and H. Xu, "An Efficient online algorithm for dynamic SDN controller assignment in data center networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2788–2801, Oct. 2017.

[58] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. R. Kompella, "ElastiCon; an elastic distributed SDN controller," in *Proc. 10th ACM/IEEE Symp. Architectures Netw. Commun. Syst.*, Oct. 2014, pp. 17–28.

[59] A. S. Muqaddas, P. Giaccone, A. Bianco, and G. Maier, "Inter-controller traffic to support consistency in ONOS clusters," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 4, pp. 1018–1031, Dec. 2017.

[60] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, "Multi-controller based software-defined networking: A survey," *IEEE Access*, vol. 6, pp. 15980–15996, 2018.

[61] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized? State distribution trade-offs in software defined networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342443

[62] B. Zhang, X. Wang, and M. Huang, "Adaptive consistency strategy of multiple controllers in SDN," *IEEE Access*, vol. 6, pp. 78640–78649, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8561291/

[63] E. Sakic and W. Kellerer, "Impact of adaptive consistency on distributed SDN applications: An empirical study," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2702–2715, Dec. 2018.

[64] S. Bhowmik, M. A. Tariq, B. Koldehofe, F. Durr, T. Kohler, and K. Rothermel, "High performance publish/subscribe middleware in software-defined networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1501–1516, Jun. 2017.

[65] T. Kohler, F. Dürr, and K. Rothermel, "ZeroSDN: A highly flexible and modular architecture for full-range distribution of event-based network control," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 4, pp. 1207–1221, Dec. 2018.

[66] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, and H. J. Chao, "Improving the performance of load balancing in software-defined networks through load variance-based synchronization," *Comput. Netw.*, vol. 68, pp. 95–109, Aug. 2014.

[67] H. Howard, M. Schwarzkopf, A. Madhavapeddy, and J. Crowcroft, "Raft refloated," *ACM SIGOPS Oper. Syst. Rev.*, vol. 49, no. 1, pp. 12–21, 2015.

[68] F. Bannour, S. Souihi, and A. Mellouk, "Adaptive state consistency for distributed onos controllers," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–7.

[69] J. Stribling, Y. Sovran, I. Zhang, I. Zhang, J. Li, M. F. Kaashoek, and R. Morris, "Flexible, wide-area storage for distributed systems with WheelFS," in *Proc. 6th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2009, pp. 43–58.

[70] O. Blial, M. Ben Mamoun, and R. Benaini, "An overview on SDN architectures with multiple controllers," *J. Comput. Netw. Commun.*, vol. 2016, Apr. 2016, Art. no. 9396525.

[71] *IEEE Standard for Local and Metropolitan Area Networks—Station and Media Access Control Connectivity Discovery*, IEEE Standard 802.1AB-2016 (Revision of IEEE Standard 802.1AB-2009), 2016, pp. 1–146.

[72] L. Ochoa-Aday, C. Cervello-Pastor, and A. A. Fernández-Fernández, "Current trends of topology discovery in OpenFlow-based software defined networks," *Int. J. Distrib. Sensor Netw.*, vol. 5, no. 2, pp. 1–6, 2015. [Online]. Available: http://upcommons.upc.edu/handle/2117/77672

[73] P. Manzanares-Lopez, J. Munoz-Gea, F. Delicado-Martinez, J. Malgosa-Sanahuja, and A. De La Cruz, "Host discovery solution: An enhancement of topology discovery in OpenFlow based SDN networks," in *Proc. 13th Int. Joint Conf. e-Bus. Telecommun. (ICETE)*, vol. 1, 2016, pp. 80–88.

[74] A. Azzouni, R. Boutaba, N. T. M. Trang, and G. Pujolle, "sOFTDP: Secure and efficient OpenFlow topology discovery protocol," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp., Cognit. Manage. Cyber World (NOMS)*, 2018, pp. 1–7.

[75] E. Rojas, J. Alvarez-Horcajo, I. Martinez-Yelmo, J. A. Carral, and J. M. Arco, "TEDP: An enhanced topology discovery service for software-defined networking," *IEEE Commun. Lett.*, vol. 22, no. 8, pp. 1540–1543, Aug. 2018.

[76] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, "A distributed algorithm for topology discovery in software-defined networks," in *Proc. Int. Conf. Pract. Appl. Agents Multi-Agent Syst.* Springer, 2016, pp. 363–367.

[77] L. Ochoa-Aday, C. Cervello-Pastor, and A. A. Fernández-Fernández, "eTDP: Enhanced topology discovery protocol for software-defined networks," *IEEE Access*, p. 1, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8642886/

[78] J. Ge, Z. Chen, Y. Wu, and Y. E, "H-SOFT: A heuristic storage space optimisation algorithm for flow table of OpenFlow," *Concurrency Comput.*, vol. 27, no. 13, pp. 3497–3509, 2015.

[79] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Rules placement problem in OpenFlow networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1273–1286, 2nd Quart., 2016.

[80] A. A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," *ACM SIGCOMM Comput. Commun. Rev.*, 2011, pp. 254–265. [Online]. Available: http://dl.acm.org/citation.cfm?id=2018466

[81] P. Song, Y. Liu, T. Liu, and D. Qian, "Controller-proxy: Scaling network management for large-scale SDN networks," *Comput. Commun.*, vol. 108, pp. 52–63, Aug. 2017.

[82] Y. Chiba, Y. Shinohara, and H. Shimonishi, "Source flow: Handling millions of flows on flow-based nodes," *ACM SIGCOMM Comput. Commun.*, vol. 41, no. 4, pp. 465–466, 2010.

[83] K. Kannan and S. Banerjee, "Compact TCAM: Flow entry compaction in TCAM for power aware SDN," in *Proc. Int. Conf. Distrib. Comput. Netw.* Springer, 2013, pp. 439–444.

[84] W. Braun and M. Menth, "Wildcard compression of inter-domain routing tables for OpenFlow-based software-defined networking," in *Proc. 3rd Eur. Workshop Softw.-Defined Netw. (EWSDN)*, Sep. 2014, pp. 25–30. [Online]. Available: http://ieeexplore.ieee.org/document/6984047/

[85] M. Rifai, N. Huin, C. Caillouet, F. Giroire, D. Lopez-Pacheco, J. Moulierac, and G. Urvoy-Keller, "Too many SDN rules? Compress them with MINNIE," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/document/7417661/

[86] H. Zhu, M. Xu, Q. Li, J. Li, Y. Yang, and S. Li, "MDTC: An efficient approach to TCAM-based multidimensional table compression," in *Proc. IFIP Netw. Conf. (IFIP Networking)*, May 2015, pp. 1–9. [Online]. Available: http://ieeexplore.ieee.org/document/7145306/

[87] S. Luo, H. Yu, and L. M. Li, "Fast incremental flow table aggregation in SDN," in *Proc. 23rd Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2014, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/document/7145306/

[88] B. Leng, L. Huang, C. Qiao, H. Xu, and X. Wang, "FTRS: A mechanism for reducing flow table entries in software defined networks," *Comput. Netw.*, vol. 122, pp. 1–15, Jul. 2017.

[89] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, p. 351, 2010.

[90] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 545–549.

[91] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the 'one big switch' abstraction in software-defined networks," in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2013, pp. 13–24. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2535372.2535373

[92] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite CacheFlow in software-defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2014, pp. 175–180. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2620728.2620734

[93] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "CacheFlow: Dependency-aware rule-caching for software-defined networks," in *Proc. Symp. SDN Res.*, 2016, pp. 6:1–6:12. doi: 10.1145/2890955.2890969.

[94] J. P. Sheu and Y. C. Chuo, "Wildcard rules caching and cache replacement algorithms in software-defined networking," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 1, pp. 19–29, Mar. 2016.

[95] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-based server load balancing gone wild into the wild: Core ideas," in *Proc. 11th USENIX Conf. Hot Topics Manage. Internet, Cloud, Enterprise Netw. Servicesworks Services (Hot-ICE)*, 2011, p. 12.

[96] C. R. Meiners, A. X. Liu, and E. Torng, "Bit weaving: A non-prefix approach to compressing packet classifiers in TCAMs," *IEEE/ACM Trans. Netw.*, vol. 20, no. 2, pp. 488–500, Apr. 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6031782

[97] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, "Source routed forwarding with software defined control, considerations and implications," in *Proc. ACM Conf. CoNEXT Student Workshop*, 2012, pp. 43–44.

[98] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, "Exploring source routed forwarding in SDN-based WANs," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 3070–3075.

[99] R. M. Ramos, M. Martinello, and C. E. Rothenberg, "SlickFlow: Resilient source routing in data center networks unlocked by OpenFlow," in *Proc. Conf. Local Comput. Netw. (LCN)*, 2013, pp. 606–613.

[100] Z. Guo, Y. Xu, M. Cello, J. Zhang, Z. Wang, M. Liu, and H. J. Chao, "JumpFlow: Reducing flow table usage in software-defined networks," *Comput. Netw.*, vol. 92, pp. 300–315, Dec. 2015.

[101] X. Jia, Y. Jiang, Z. Guo, and Z. Wu, "Reducing and balancing flow table entries in software-defined networks," in *Proc. Conf. Local Comput. Netw. (LCN)*, 2016, pp. 575–578.

[102] X. Jia, Q. Li, Y. Jiang, Z. Guo, and J. Sun, "A low overhead flow-holding algorithm in software-defined networks," *Comput. Netw.*, vol. 124, pp. 170–180, Sep. 2017. doi: 10.1016/j.comnet.2017.06.009.

[103] X. Dong, Z. Guo, X. Zhou, H. Qi, and K. Li, "AJSR: An efficient multiple jumps forwarding scheme in software-defined WAN," *IEEE Access*, vol. 5, pp. 3139–3148, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7862203/

[104] N. Kitsuwan, S. Ba, E. Oki, T. Kurimoto, and S. Urushidani, "Flows reduction scheme using two MPLS tags in software-defined network," *IEEE Access*, vol. 5, pp. 14626–14637, 2017.

[105] R. Carpa, O. Gluck, and L. Lefevre, "Segment routing based traffic engineering for energy efficient backbone networks," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst. (ANTS)*, Dec. 2014, pp. 1–6.

[106] L. Davoli, L. Veltri, P. L. Ventre, G. Siracusano, and S. Salsano, "Traffic engineering with segment routing: SDN-based architectural design and open source implementation," in *Proc. Eur. Workshop Softw. Defined Netw., EWSDN*, 2015, no. 1, pp. 111–112.

[107] A. Sgambelluri, F. Paolucci, A. Giorgetti, F. Cugini, and P. Castoldi, "Experimental demonstration of segment routing," *J. Lightw. Technol.*, vol. 34, no. 1, pp. 205–212, Jan. 1, 2016.

[108] M.-C. Lee and J.-P. Sheu, "An efficient routing algorithm based on segment routing in software-defined networking," *Comput. Netw.*, vol. 103, pp. 44–55, Jul. 2016. [Online]. Available: http://www.sciencedirect/science/article/pii/S1389128616300871

[109] J.-P. Sheu and Y.-C. Chen, "A scalable and bandwidth-efficient multicast algorithm based on segment routing in software-defined networking," in *Proc. IEEE Int. Conf. Commun.*, May 2017, pp. 1–6.

[110] C. Filsfils, D. Cai, S. Previdi, W. Henderickx, R. Shakir, D. Cooper, F. Ferguson, T. LaBerge, S. Lin, B. Decraene, and L. Jalil, *Interconnecting Millions of Endpoints With Segment Routing*, document rfc8604, IETF, 2018. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-filsfils-spring-large-scale-interconnect-09

[111] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, *Segment Routing Architecture*, document rfc8402, IETF, 2018. [Online]. Available: https://tools.ietf.org/html/rfc8402

[112] C. Filsfils, S. Previdi, J. Leddy, S. Matsushima, and D. Voyer, *IPv6 Segment Routing Header (SRH)*, document draft-ietf-6man-segment-routing-header-21, IETF, 2018. [Online]. Available: http://tools.ietf.org/html/draft-ietf-6man-segment-routing-header

[113] A. Sgambelluri, A. Giorgetti, F. Cugini, G. Bruno, F. Lazzeri, and P. Castoldi, "First demonstration of SDN-based segment routing in multi-layer networks," in *Proc. Opt. Fiber Commun. Conf. Exhib. (OFC)*, Mar. 2015, pp. 1–3.

[114] P. L. Ventre, M. M. Tajiki, S. Salsano, and C. Filsfils, "SDN architecture and southbound APIs for IPv6 segment routing enabled wide area networks," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 4, pp. 1378–1392, Oct. 2018.

[115] L. Zhang, R. Lin, S. Xu, and S. Wang, "AHTM: Achieving efficient flow table utilization in software defined networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2014, pp. 1897–1902.

[116] L. Zhang, S. Wang, S. Xu, R. Lin, and H. Yu, "TimeoutX: An adaptive flow table management method in software defined networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–6.

[117] H. Liang, P. Hong, J. Li, and D. Ni, "Effective idle_timeout value for instant messaging in software defined networks," in *Proc. IEEE Int. Conf. Commun. Workshop (ICCW)*, Jun. 2015, pp. 352–356.

[118] M. Lu, W. Deng, and Y. Shi, "TF-IdleTimeout: Improving efficiency of TCAM in SDN by dynamically adjusting flow entry lifecycle," in *Proc. IEEE Int. Conf. Syst., Man, (SMC)*, Oct. 2017, pp. 2681–2686.

[119] Y. Liu, B. Tang, D. Yuan, J. Ran, and H. Hu, "A dynamic adaptive timeout approach for SDN switch," in *Proc. 2nd IEEE Int. Conf. Comput. Commun. (ICCC)*, Oct. 2017, pp. 2577–2582.

[120] Z. Guo, R. Liu, Y. Xu, Y. Gushchin, A. Walid, and H. J. Chao, "STAR: Preventing flow-table overflow in software-defined networks," *Comput. Netw.*, vol. 125, pp. 15–25, Oct. 2017. doi: 10.1016/j.comnet.2017.04.046.

[121] Z. A. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt, and G. Noubir, "Application-awareness in SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 487–488, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2534169.2491700

[122] L. He, C. Xu, and Y. Luo, "VTC: Machine learning based traffic classification as a virtual network function," in *Proc. ACM Int. Workshop Secur. Softw. Defined Netw.; Netw. Function Virtualization*, 2016, pp. 53–56. doi: 10.1145/2876019.2876029.

[123] T. Kim, K. Lee, J. Lee, S. Park, Y.-H. Kim, and B. Lee, "A dynamic timeout control algorithm in software defined networks," *Int. J. Future Comput. Commun.*, vol. 3, no. 5, pp. 331–336, 2014.

[124] L. Kuang, L. T. Yang, X. Wang, P. Wang, and Y. Zhao, "A tensor-based big data model for QoS improvement in software defined networks," *IEEE Netw.*, vol. 30, no. 1, pp. 30–35, Jan./Feb. 2016.

[125] P. Bull, R. Austin, and M. Sharma, "Pre-emptive flow installation for Internet of Things devices within software defined networks," in *Proc. 3rd Int. Conf. Future Internet Things Cloud*, 2015, pp. 124–130. doi: 10.1109/FiCloud.2015.87.

[126] S.-C. Chao, K. C.-J. Lin, and M.-S. Chen, "Flow classification for software-defined data centers using stream mining," *IEEE Trans. Services Comput.*, vol. 12, no. 1, pp. 105–116, Jan./Feb. 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7549048/

[127] F. Chen and X. Zheng, "Machine-learning based routing pre-plan for SDN," in *Proc. Int. Workshop Multi-Disciplinary Trends Artif. Intell.* Springer, 2015, pp. 149–159.

[128] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv, "Control plane of software defined networks: A survey," *Comput. Commun.*, vol. 67, pp. 1–10, Aug. 2015. doi: 10.1016/j.comcom.2015.06.004.

[129] Z. Liu, D. Gao, Y. Liu, H. Zhang, and C. H. Foh, "An adaptive approach for elephant flow detection with the rapidly changing traffic in data center network," *Int. J. Netw. Manage.*, vol. 27, no. 6, p. e1987, 2017. doi: 10.1002/nem.1987.

[130] T. Hu, P. Yi, Z. Guo, J. Lan, and Y. Hu, "Dynamic slave controller assignment for enhancing control plane robustness in software-defined networks," *Future Gener. Comput. Syst.*, vol. 95, pp. 681–693, Jun. 2019. 10.1016/j.future.2019.01.010.

[131] S. Gao, Y. Zeng, H. Luo, and H. Zhang, "Scalable control plane for intra-domain communication in software defined information centric networking," *Future Gener. Comput. Syst.*, vol. 56, pp. 110–120, Mar. 2016. doi: 10.1016/j.future.2015.10.017.

[132] Y.-W. Ma, J.-L. Chen, Y.-H. Tsai, K.-H. Cheng, and W.-C. Hung, "Load-balancing multiple controllers mechanism for software-defined networking," *Wireless Pers. Commun.*, vol. 94, no. 4, pp. 3549–3574, 2017.

[133] S. Wang, J. Zhang, T. Huang, T. Pan, J. Liu, and Y. Liu, "Flow distribution-aware load balancing for the datacenter," *Comput. Commun.*, vol. 106, pp. 136–146, Jul. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366417303043

[134] L. Li and Q. Xu, "Load balancing researches in SDN: A survey," in *Proc. 7th IEEE Int. Conf. Electron. Inf. Emergency Commun. (ICEIEC)*, Jul. 2017, pp. 403–408.

[135] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 7–12, Oct. 2013. doi: 10.1145/2534169.2491193.

[136] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-jacobson, "Pratyaastha: An efficient elastic distributed SDN control plane," in *Proc. HotSDN*, 2014, pp. 133–138.

[137] C. Wang, B. Hu, S. Chen, D. Li, and B. Liu, "A switch migration-based decision-making scheme for balancing load in SDN," *IEEE Access*, vol. 5, pp. 4537–4544, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7883881/

[138] T. Hu, P. Yi, J. Zhang, and J. Lan, "A distributed decision mechanism for controller load balancing based on switch migration in SDN," *China Commun.*, vol. 15, no. 10, pp. 129–142, Oct. 2017.

[139] P. Lin, J. Bi, and H. Hu, "ASIC: An architecture for scalable intra-domain control in OpenFlow," in *Proc. 7th Int. Conf. Future Internet Technol.*, 2012, pp. 21–26. doi: 10.1145/2377310.2377317.

[140] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "BalanceFlow: Controller load balancing for OpenFlow networks," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Intell. Syst. (CCIS)*, vol. 2, Oct./Nov. 2012, pp. 780–785.

[141] F. Cimorelli, F. D. Priscoli, A. Pietrabissa, L. R. Celsi, V. Suraci, and L. Zuccaro, "A distributed load balancing algorithm for the control plane in software defined networking," in *Proc. 24th Medit. Conf. Control Autom. (MED)*, Jun. 2016, pp. 1033–1040.

MOHAMMED ALSAEEDI received the B.S. degree in computer science from King Abdulaziz University and the M.S. degree in computer networks from the King Fahd University of Petroleum and Minerals. He is currently pursuing the Ph.D. degree with Universiti Teknologi Malaysia. His research interests include the Internet-of-Things data analytics and machine learning for smart cities, real-time and distributed systems, and software-defined networking.

MOHD MURTADHA MOHAMAD received the bachelor's degree in computer from Universiti Teknologi Malaysia (UTM) and the master's degree in embedded system engineering and the Ph.D. degree in electrical engineering from Heriot-Watt University. He is currently a Senior Lecturer with UTM, where he is also the Deputy Director (Application Development Management) of the Centre for Information and Communication Technology.

ANAS A. AL-ROUBAIEY received the B.S. degree in computer engineering from the Arab Academy for Science and Technology, Alexandria, Egypt, in 2001, and the M.S. degree in computer networks and the Ph.D. degree in computer science and engineering from the King Fahd University of Petroleum and Minerals, Saudi Arabia, in 2009 and 2015, respectively. He was a Teaching Assistant with Taiz University, until 2004. He is currently a Researcher with the King Fahd University of Petroleum and Minerals. His research interests include intrusion detection systems, middleware and systems integration, real-time and distributed systems, wireless sensor and actuator networks, and energy-aware and power harvesting mechanisms in wireless sensor networks.