Received May 2, 2019, accepted May 19, 2019, date of publication May 30, 2019, date of current version June 18, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2919957

A Survey on Autonomic Provisioning and Management of QoS in SDN Networks

AHMED BINSAHAQ[1], TAREK R. SHELTAMI[1], AND KHALED SALAH[2]
[1]Computer Engineering Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia
[2]Electrical and Computer Engineering Department, Khalifa University, Abu Dhabi, United Arab Emirates

Corresponding author: Ahmed BinSahaq (ishaq1900@gmail.com)

This work was supported by the King Fahd University of Petroleum and Minerals for this Research.

ABSTRACT In today's Internet, killer network services and applications, such as video and audio streaming, network storage, and online video games, are pushing the network infrastructure resources to the edge. By design and for the most part, the Internet is the best offer delivery ecosystem with little or no end-to-end quality-of-service (QoS) guarantees. Even, frameworks, such as *IntServ* and *DiffServ* that were designed and implemented to provide QoS guarantees, still fail to solve this problem at a wide scale. Software-defined networking (SDN) is a fast emerging networking paradigm that promises to provide end-to-end QoS guaranteeing by offering greater network flexibility, abstraction, control, and programmability to network resources. In this paper, we review, survey, and discuss the current state of the art on QoS provisioning in the area of SDN, with respect to applying the concept of autonomic computing (AC) to automatically support, provision, monitor, and maintain QoS requirements. This paper includes in-depth classification, taxonomy, and comparative analysis for the autonomic-based QoS provisioning in accordance with the famous influential and widely adopted the monitor-analyze-plan-execute-knowledge (MAPE-K) IBM architectural model for autonomic computing.

INDEX TERMS Autonomic computing, autonomous systems, automation, autonomic networking, SDN, QoS.

I. INTRODUCTION

In 2017, Cisco reported that the global annual IP traffic reached 1.2 Zettabytes in 2016 and it is expected to reach 3.3 Zettabytes per year by 2021 [2]. In addition 82% of the consumers' Internet traffic will be IP video traffic by 2021. From the overall Internet video traffic, 13% will be live video traffic [2]. Also, the ratio of Internet-connected devices will be three times the global population.

With this pressure on today's Internet, service providers were between two choices, either put in extra resources into their networks, which adds extra costs, or apply strict policies, which may not satisfy their customers. In addition, they are obligated to provide services with certain quality as it is agreed to in the Service Level Agreement (SLA). However, the massive increase in the numbers of Internet-connected devices (e.g. user's mobile devices, servers, Things, etc) is promising a lot of money at stake for service providers from now and upon [2]. At that point, providing services with certain QoS guarantee while efficiently utilizing the network resources becomes a challenging task for many service providers or network operators [3].

There are two models proposed to support QoS over the Internet, namely: Integrated Services (IntServ) [4] and Differentiated Services (DiffServ) [5]. The earliest one is designed to reserve sufficient resources along the path that the service packets use. The work is evolved into a design of a signaling protocol called the Resource Reservation Protocol (RSVP) [6] that was designed to work over IP protocol. However, it has a scalability problem since it needs to keep state on each network node about each going traffic flow. In *DiffServ*, packets are tagged to differentiate between the service level they require (i.e. VoIP requires a 150ms maximum delay) or receive. Therefore, each packet is treated differently based on its tag or class and the QoS policy is configured on each node. By doing that, different QoS classes can be defined for variant services. *DiffServ* is designed to provide end-to-end QoS support by assuring QoS application on each intermediate node within the network.

DiffServ cannot provide guarantees like those offered by *IntServ*, and suffers from dynamic changes of application requirements. Moreover, the necessity to reconfigure each

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

node within the network to reflect a change in QoS policy or new SLA requirement causes a huge headache for network operator probability of mistakes is high. The usage of automated tools could be a solution. However, due to the diversity of network vendors and the complexity of the current architecture (i.e. traditional networks with different types of equipment) makes it worse. To alleviate those problems, a new architecture that supports generality and programmability of network elements is defined by ONF (Open Networking Foundation) [8]. By separating the control for the data planes it provides agility and flexibility for the network operator. The new architecture is called Software Defined Networking [9]. SDN is built based on flow-based programming in which network devices are programmed dynamically and on the fly by installing flow rules to forward packets. The control logic is centralized in a controller node, which helps network operator to get global view and control of the whole network. This allows operator to implement applications that adjust networks behavior to meet new needs or manage themselves [10]. The simplicity of service and network provisioning was the most important driver for investments in SDN deployment by many service providers [11], [12] to achieve organization goals [13].

Support of QoS is extensively addressed in the literature for traditional networks. However, the emergence of SDN opens the door to tackle that problem again. Promising features, such as the decoupling of network control from forwarding, also the global view of the whole network, gives SDN advantage to support QoS.

This survey article is about QoS in SDN with focus on the autonomic support of QoS in SDN-based networks. Readers can find a comprehensive survey about the SDN architecture in [18], [19]. In the literature, there are two surveys about QoS in SDN [21] and [22]. In [21], the authors discuss the concept of QoS and how it can be improved via SDN. They focused on four main fields, namely; resources reservation, flow-based routing, queue management, and policy enforcement. The work in [22] is more comprehensive than the previous one. The authors discussed more recent research work done aforementioned four fields. Beside of that, they focused on the user experience of QoS and network monitoring as another prominent field to improve QoS in SDN. In this work, we focus on QoS management and provisioning research works that leverage the functionality of Autonomic Computing(AC) [23] in QoS support within SDN. The authors in [24] applied the concept of Autonomic Computing on existing network architectures to identify the characteristics of Autonomic Networking within. However, SDN is not considered by them since they focused on projects implemented and tested by that time. For the purpose of generality, we had to analyze the SDN features against Autonomic Computing properties. The authors in [15] discussed the autonomy of SDN communication in conjunction with NFV (Network Functions Virtualization). They focused on subjects such as testing and integration network functions. However, the support of QoS is not discussed by the previously mentioned work. The authors in [16] went through network monitoring in SDN. Similarly, the authors in [17] discuss the challenges in measurement collections within SDN. Part of this survey is focusing on SDN network monitoring as a function of autonomic QoS provisioning in SDN based networks. The contribution of this survey can be summarized as follows:

- This survey article provides a comprehensive overview of SDN architecture with autonomic support for QoS.
- The article reviews, summarizes, and discusses the current state-of-the-art on QoS provisioning from autonomy perceptive.
- In this article we try to answer the question, does current literature contribute to support autonomic management of QoS in SDN?
- The article provides thorough classification, taxonomy, and comparative analysis for autonomic-based QoS provisioning in accordance with the popular and widely adopted Monitor-Analyze-Plan-Execute-Knowledge (MAPE-K) architectural model for autonomic computing.

The remainder of this survey article is organized as follows. In II we discuss the concept of Autonomic Computing and Autonomic networking. In Section III, we give a brief background on SDN, analyze the SDN architecture and discuss its features as Autonomic System. Next, we discuss related work to support QoS provisioning and management based on SDN in Section IV. Then, we discuss QoS requirements specification in Section V. In Section VI, we discuss and introduce future research directions. Lastly, we conclude our survey article in Section VII.

II. AUTONOMIC COMPUTING

The management of complex systems is a challenging task that requires well-skilled team. The cost scales with the complexity of such systems. Therefore, the design of a self-managed system reduces such costs. Moreover, it adapts and reacts to its environments changes with no or little intervention of humans, which is the vision of Autonomic Computing (AC) [23]. The main goal of Autonomic Computing is to design a self-managed system that operates based on high-level rules or policies formed by the system administrator(s) [23]. Those rules usually follow IF-THEN or event-condition-action form [25]. This concept is inducted from the human nervous system which operates autonomically to control the human body [26], [27] (e.g. blood sugar, temperature, heartbeats control) without intervention from the outside. The delegation of system management tasks(functions), frees human to focus on business goals that increase organization profit.

A. SELF-* PROPERTIES OF AUTONOMIC COMPUTING (AC)

Autonomy of a system is a characteristic of the way it controls its functions to achieve the self-management goal. IBM characterized the self-management of an autonomic

TABLE 1. Abbreviation/acronyms.

API	Application Programming Interface
CDPI	Controller-Data Plane Interfaces
CSP	Constraint Shortest Path
DBMS	Data Base Management System
DPI	Deep Packet Inspection
DPID	Data Path Identifier
DRL	Deep Reenforcement Learning
DSCP	Differentiated Services Code Point
FIFO	First In First Out
HTB	Hierarchical Token Bucket
IETF	Internet Engineering Task Force
IP	Internet Protocol
IXP	Internet Exchange Point
KPI	Key Performance Indicator
LARAC	Lagrange Relaxation based Aggregated Cost
LLDP	Link Layer Discovery Protocol
MCSP	Multiple Constrained Shortest Path
MOS	Mean Opinion Square
NFV	Network Functions Virtualization
ONF	Open Networking Foundation
OVS	OpenVSwitch
OVSDB	Open vSwitch Database Management Protocol
PSNR	Peak Signal to Noise Ratio
QoS	Quality of Service
RED	Random Early Detection
REST	REpresentational State Transfer
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SBI	South-Bound Interfaces
SDN	Software Defined Networking/Software Defined Network
SFQ	Stochastic Fairness Queuing
SLA	Service Level Agreement
SLO	Service Level Object
SVC	Scalable Video Coding
TCAM	Ternary Content Addressable Memory
ToS	Type of Service
VC	Virtual Circuit
VLAN	Virtual Local Area Network
VoIP	Voice over IP
vSDN	virtual Software Defined Network

system (i.e., that follows the Autonomic Computing principle) in four self-* properties namely; *self-configuration*, *self-healing*, *self-optimization* and *self-protection* [23], [27]–[31].

- **Self-configuration**. An autonomic system should be able to configure itself in response to changes in its environment and according to high-level policies that define what the system should do.
- **Self-healing**. An autonomic system should detect problems and tries to recover or correct itself (if possible) by adjusting or reconfiguring its components which makes its daily operations fault-tolerance.
- **Self-optimization**. An autonomic system should maximize utilization of its resources and monitoring its performance against an ideal case. This process should be guided through high-level policies that may define a utility function (i.e. utility based management) for the system where some tasks are prioritized over others [32]. It also may need to acquire more resources or evacuate others to achieve optimal state (its objectives).
- **Self-protection**. An autonomic system should detect internal or external threads and protect its resources through appropriate defense actions that assure its security and privacy.

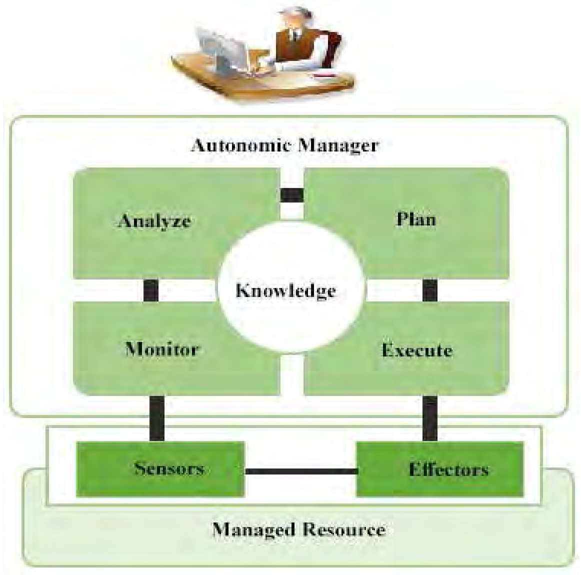


FIGURE 1. IBM's MAPE-K reference model [24].

In order to achieve those four self-* properties, the autonomic system needs to be able to monitor itself, its environment and adjust itself through reconfiguration of its components. This adjustment can be triggered by either a threat, an optimization purpose, or a change in high-level policies. This is referenced in a model by IBM [27] called MAPE-K (i.e. short for Monitor, Analyze, Plan, Execute and Knowledge), which is performed and controlled by the main component of the autonomic system, called "Autonomic Manager".

B. MAPE-K

Fig. 1 illustrates the MAPE-K reference architecture for autonomic computing and its key functional components:

- **Monitor**. It collects data from the managed resources or elements through touch-points or interfaces (i.e APIs). Data could be events generated by the managed element, its state, or its configuration. The size of such data could be large, therefore, it may need to be filtered and structured in a way to be ready for analysis.
- **Analyze**. It analyzes collected data by the monitor function and observe the system state. It may detect that some policies are not met by the system or the system performance is not as it should be. Therefore, a change is issued to the plan function with all necessary details.
- **Plan**. Based on the requirement of changes, a plan to change may be formed to the existing state or configuration of the system. This plan defines the work to be done in a suitable form and sends it to the execute function.
- **Execute**. This function performs a series of actions on the system to reflect changes required by the plan function. An action could be a change to the system configuration or more resources are occupied.

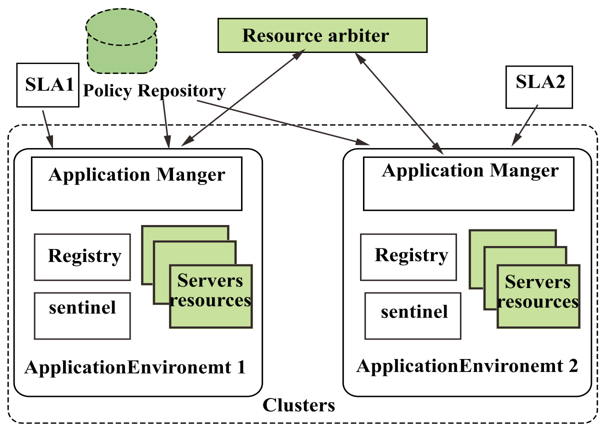


FIGURE 2. Unity architecture [24], [33].

- **Knowledge**. Data collected or generated by the above functions such as system state, configurations, policies, plans, actions' history is stored in a knowledge source.

From Fig. 1, beside the MAPE-K control loop, it shows components of such autonomic system (or element): 1) autonomic manager, which execute the MAPE-K control loop, 2) managed resources such as servers, network devices (e.g. switches, routers, firewalls, load balancers), 3) touch-points that connects autonomic manager with managed resources, 4) sensors and effectors. Sensors could be agents for collecting monitoring data and send it to the monitor function. Similarly, effectors could be agents for executing commands sent by the execute function to enforce a plan (e.g. optimization plan). This structure is common in autonomic system designs. Administration layer is above all of those, which defined general rules or policies to guide the autonomy functioning of the system and to serve the business goals.

Fig. 2 shows an architecture for one of those systems that tried to tackle the self-management by the design of an autonomic system. This architecture is called Unity [33]–[35] and it been carried out at IBM's Thomas J. Watson Research Center. In Unity, the system consists of autonomic elements each is responsible for its behavior such as controlling its resources or its internal operations. Also its external behavior with other autonomic elements (i.e. through defined interface only). By doing that, the self-management of the whole system is achieved. Authors used the achievement of SLA as the indication of the system success or its performance (i.e. as an evaluation scenario). The system consists of one or more *application managers* (i.e. applications 1 and 2) each responsible for the application's environment (e.g. the web application). It predicts the changes in the application utility, whether its resources are increased or decreased. A set of servers are representing a pool of resources in which the other system's element, *arbiter*, controls. It can reassign resource from one application to another. This element performs the optimization of resources utilization to increase the overall system utility and meeting requirements of each application SLA. Each application's environment contains a *repository* that holds state information, which can be polled by the *arbiter* or the system administrator. Also, another element called *sentinel* it responsible to monitor the services functionality. If any fault or a service stopped, it notifies the interesting entities (e.g. administrators, or another element). This is part of the self-healing property that Unity peruse. Besides polling the *repository*, an administrator can interface another element called *policy repository* for defining the general rules or utility function that guide the system behavior. This repository is also used by the *arbiter* while it optimizes the resource allocation process. The *arbiter* asks each *application manager* about the expected utility of its environment when certain amount of resources is assigned to. Based on that predicted utility, a utility-based optimization for resources assignment is performed by the *arbiter* element. Despite the system aimed to achieve all four autonomic computing attributes, however, the self-protecting property was not clear.

C. AUTONOMIC NETWORKING

Similarly and with the same Autonomic Computing concept, a design for autonomic network architecture will solve many management problems (i.e. due to network complexity, cost, errors) and frees experts for innovation tasks. Therefore, an autonomic network architecture should provide self-management (i.e. the four self-* properties) for network owners. The task of achieving autonomy for network management is more complex due to the high heterogeneity of network elements and those protocols working upon.

There is a big difference between *autonomicity* and *automaticity* in network management. *Automaticity* means automating routine tasks that are handled by the networks, either on the element level such processing of packets within network router or at the administration level by using tools or scripts to automatically collecting data from those elements through protocols such as SNMP [36]. The latter does not involve self-optimization of the network performance to achieve the best utility drawn by high-level rules or goals which is the purpose of *autonomicity*. In current IP networks, there are some forms of *autonomicity*. For instance, routing protocols such as OSPF [37] makes routers work together to autonomically coordinate packets' routing process. However, those network elements still need external configuration by network administrators (i.e. especially if the environment changed).

Many network architectures proposed to overcome management complexity or to introduce autonomic networking. ANA (Autonomic Network Architecture) [38], [39] project was one of those proposed in the literature. It aimed at providing a communication abstraction for different heterogeneous network styles. An ANA node contains one or more *Functional Block*, *Information Channel*, and *Information Dispatch Point*. Instead of using protocol stack such as the one in OSI model, one or more node's functional block(s) perform the task or the service required. And for information or data transmission, an information's channel is used between those interesting endpoints. Either between nodes within the same context which they called compartment (or another compartment's nodes). Compartment defines its own communication policies, membership, trust, addressing, routing and policies to interact with other compartments. The address resolution is defined by each compartment. It defines an information dispatch point to reach its nodes. Information's channels are bounded to those dispatch point to allow flows of information between interesting compartment or nodes within. Also authors in [40] proposed a management architecture to support self-configuration and self-optimization in current IP networks. They divided the network management into two layers, *Objective definition layer* and *Objective achievement layer*. In the first one, network operators define requirements of the system at the highest level which represent the business view of the system. Then those requirements are transformed into *network utility functions (NUFs)* by a main system component called *Objective Definition Point*(ODP). Beside network operators, at that management level, experts provide management guidelines or strategies. NUFs represents the polices to describe the network performance at the system view (e.g. optimization functions). Those NUFs and experts strategies are forwarded to another component called *Global Definition Point*(GDP). This component analyzes the NUFs and chooses the best expert strategies to achieve them. The results are a set of goal policies or specifications. The network infrastructure is divided into small domains managed by a coordinator called *Domain Goal Distribution Point*(DGDP). The GDP distribute those NUFs and goal policies to all DGDPs. DGDP analyze them and identifies the appropriate behavioral policies to achieve those goal policies. These behavioral policies are targeting entities in the second management layer called *goal achievement points*. They represent the autonomic managed entities in the system (i.e. they could be router, switch, gateway). They perform the MAPE-K control to manage their behavior and interact with their context guided by behavioral policies received from DGDPs. In that system, human interaction with the system in two ways, network operator who define system requirements and experts who provide knowledge expressed in guidelines or strategies.

In [24], the authors did a survey on similar exiting networking architectures which tries to apply autonomy principles (i.e. early discussed in Section II-A) to achieve self-management within networking. Authors also provided qualitative and quantitative criteria to evaluate and compare between such architectures. They mentioned that autonomic network can be evaluated quantitatively, besides other criteria, through QoS metrics since they can be used to measure the performance of an autonomic network. We believe so, however, we think of the network as a system that provides a service of data transportation for its users and owners. Therefore, management of QoS for services or applications that use that system should be autonomic. Such management task could extensively benefit from the underlying autonomic network. Also, the authors did not consider new network architectures such as SDN and the opportunity in it to support autonomy in networking.

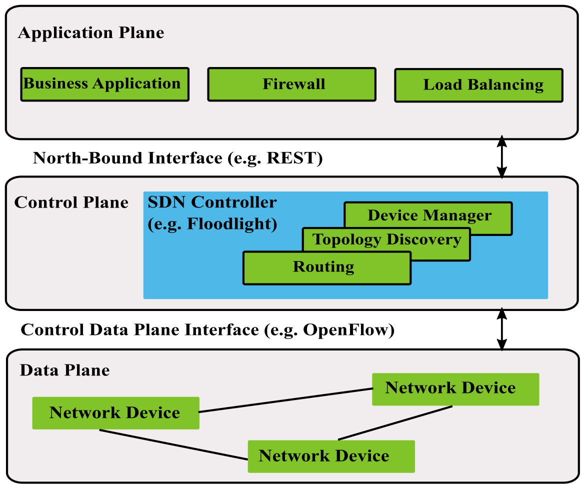


FIGURE 3. Software defined networking architecture.

III. SOFTWARE DEFINED NETWORKING

SDN is a new network architecture that decouples control plane from data plane differently than the current network architecture [9]. The idea behind it was to use flow tables inside network devices and a standard interface for flow table configuration, management, and manipulation. The standard protocol to accomplish this task was called OpenFlow [14]. Those flow tables contain rules that match packet header fields and actions to perform on those matched packets [14]. OpenFlow protocol has become the de-facto protocol for SDN by which a central server controls network devices through OpenFlow primitives to install and delete flow table entries.

Fig. 3 shows an overview of the SDN architecture. Control and data planes are physically separated differently from today's widely deployed network architecture. In which the network device (e.g., router) controls the forwarding tasks through routing protocol implemented within it. In the SDN, there are three separate planes:

- **Data Plane**. This plane consists of OpenFlow compliant forwarding devices of either physical or virtual switches. They can be considered as dump forwarding devices with OpenFlow firmware installed on. OpenVswitch [41] is an example of such platforms. Many vendors of network devices made their new products compatible with OpenFlow protocol as Table 2 shows. The communication between SDN controller and the switch is performed using a South-Bound Interfaces SBI) or Controller-Data Plane Interfaces (CDPI) by which the controller can install, modify or delete flow tables rules.
- **Control Plane**. In this plane, the control of the whole network devices resides. Many other functionalities such

TABLE 2. Sample of OpenFlow compatible switches.

Ref	Switch	H/S	Management Protocol	Provider	QoS Support
[41]	OpenVSwitch	S	OpenFlow (1.1, 1.2), OVSDB	Linux Foundation	Traffic Queuing and Shaping
[42]	ofsoftswitch13	S	OpenFlow (1.3)	Ericsson Innovation Center	Traffic Queuing and Shaping
[43]	Indigo	S	OpenFlow (1.0)	Big Switch Networks.	–
[44]	Arista 7150	H	OpenFlow (1.0), SNMP	Arista	Traffic Queuing and Shaping
[45]	NEC PF5240 Switch	H	OpenFlow (1.0,1.3.1)	NEC	Rate limiting, bandwidth control
[46]	Pica8 P-3297	H	OpenFlow (1.4)	Pica8	–

TABLE 3. Sample of open source SDN controllers.

Ref	Controller	Language	Origin	License	Architecture	South-Bound API	*North-Bound API*	GUI	QoS support
[47]	NOX	C++/ Python	Nicira	GPL	Centralized	OF v1.0	REST API	–	–
[52]	Floodlight	Java	Big Switch Networks	Apache	Centralized	OF v1.0-v1.4	REST API	Web-based	Metering, latency based-routing
[48]	POX	Python	Nicira	Apache	Centralized	OF v1.0	REST API	Yes	–
[53] [54]	RYU	Python	NTT Communications	Apache	Centralized	OF v1.0-v1.5, NetConf, OF-Config	–	–	Metering, Rest-based rate configuration
[49]	Maestro	Java	Rice University	LGPL	Centralized	OF v1.0	REST API	–	–
[55]	Open Daylight	Java	The Linux Foundation	Apache	Distributed	OF v1.0, 1.3, SNMP, NETCONF, OVSDB, and many others	REST API	Web-based	OVSDB-based queue configuration, Metering
[50] [51]	Beacon	Java	Standford University	GPLv2	Centralized	OF v1.0.1	REST API	Web-based	–
[56]	ONOS	Java	Open Networking Lab	Apache	Distributed	OF v1.0,1.3,1.4, SNMP, NETCONF, OVSDB, and many others	REST API	Web-based	Metering, OVSDB-based queue configuration
[57] [58]	IRIS	Java	IRIS Research Team of ETRI	Apache	Distributed	OF v1.0.1-1.3.2	REST API	Web-based	–

as forwarding, topology discovery, fault detection and many others could be performed in this plane. SDN controllers are the embodiment of the SDN architecture. Different implementations are proposed and deployed for acquiring SDN as shown in Table 3. The controller provides an abstraction of the underlying network infrastructure and services to the network applications. Many network applications such as load balancer can obtain information through through North-Bound Interfaces (NBI) from the controller. For instance, a load balancer can retrieve information about the status of the network switches or links since they are maintained and monitored by the controller. The main drawback of the SDN is that the control functionality is centralized which makes it a single point of failure. However, many researchers have proposed solutions to overcome such problem by using multiple distributed controllers [20].

- **Application Plane**. In this plane, network and business applications reside. Programmability of the SDN architecture allows these applications to interact with the controller without concerning of complexity of the network due to the abstract view that SDN provides and availability of network data through northbound APIs. Which separate complex network configuration and from application developments.

A. AUTONOMY OF SDN

In legacy network architectures, network management is centralized while the control is distributed and implemented within network devices (i.e., through routing protocols, spanning tree). The SDN architecture has the concept of removing the control from network elements and put it in a logical central place. This is the second concept that SDN was intended to achieve which the abstraction of the network view and control that allows multiple layers to coexist without intervening with each others' resources and control. This decoupling of control and data flow, allows the concept of network programmability to be a reality since the decision is maintained centrally by the controller. Therefore, a developed software can control the behavior of its network for its goals. All on the fly just by installing OpenFlow rules within those

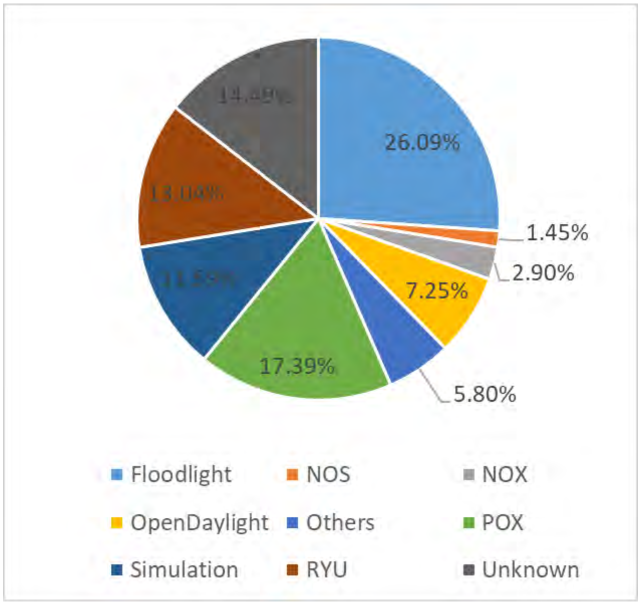


FIGURE 4. Distribution of used SDN controllers in reviewed works.

managed elements. Therefore, the role of the SDN controller is the most important or as it been the operating system of the network to support autonomic networking using the SDN architecture.

As discussed previously and in [24], achieving autonomic networking is a dream for network providers and operators. Moreover, it will be an advantage of any new architecture to support autonomic management of its components. SDN is a promising architecture due to its neutral design that is built based on the decoupling of control and data planes. By providing simple primitives through which applications can program the network and changes its behavior accordingly [67]. It allows SDN-based networks to change its configuration just by replacing flow rules within network switches (*Self-Configuration*). For instance, in response to changes in traffic load or due to a link failure.

Central control of the forwarding decision allows the controller to compare every connection request or flow initiation against security policies. For instance, when new users want to connect to an external website, that domain or IP can be tested within the controller against a set of security rules by which either that flow is admitted or rejected. The controller creates flow rules within switches that allow/deny that flow's packets within the network (*self-protecting*). Also, network operators can configure those switches to behave similarly to mini-firewalls and may steer specific type of traffic to another system (i.e., such as IDS) either for more inspection [68] or detection of possible threats [69]–[72]. Many SDN controllers already implement firewall application within their distribution such as in OpenDaylight and Floodlight controllers.

The global view of the network provided by the SDN controller allows optimizing the whole network traffic according to specific objectives (*self-optimization*). For instance, a load balancing application can use that information to distribute flows along different paths (i.e., through the controller) [73]. Topology management is a critical module in most of SDN controller. It builds a view about the status of network elements and links between them. For instance, protocols such as Link Layer Discovery Protocol (LLDP) can be used for failure detection. In response to such incident (i.e., or even a threat detection), SDN controller can reconfigure those devices to use other available paths instead of those failed ones with little service interruption (*self-healing*) [74].

B. QoS SUPPORT IN OpenFlow

By design, SDN provides network operators with granular control over traffic flows. Forwarding rules installed within the flow table targeting a specific flow(s) using packet header information (i.e., 2nd-level, 3rd-level or 4th-level of TCP/IP stack). In its early specifications, OpenFlow allows actions to be taken on a flow's packet after it matches a rule within the flow table. In OpenFlow 1.0 [76], an *enqueue* action is defined by which a packet can be attached to one of the output port queues. It allows the controller to assign different flow traffic such as those to belong to VoIP service into a specific queue (i.e., for sensitive delay requirements). This feature is limited compared to what is already defined in the traditional network devices. Moreover, queues configuration task is assigned to another protocol that is later defined called *OF-Config* [75]. In OpenFlow 1.1 [77], the ability of multi-level VLAN/MPLS and traffic classes (i.g. ToS/DSCP) matching was added. Also, an action was specified to add, remove or modify those labels (or tags). Moreover, group tables are introduced which allows flows aggregated action to be performed. A flow entry(s) in the flow table can point to an entry in the group table for more processing or to aggregate statistics separately than forwarding rules. In OpenFlow 1.2 [78], the controller becomes able to query switches for queues configuration such as max-rate. Also, it becomes possible to attach queues to different ports. In OpenFlow 1.3 [79], the ONF working group guidelines offer to implement rate limiting by using meter tables with its structure shown in Fig. 7. Meter table contains meter entries each one is identified with a 32bit unsigned integer identifier. A meter can be attached to flow entry. Counters within meter to count packets that been processed by the meter. For rate limiting, meter bands are used to compute the rate for flows attached to that meter aggregately. Band type specifies what to do such as marking the packet with DSCP value or drop it (i.e., by adding it to the action or instruction set to execute). Therefore, if the current flow(s) rate is below configuration, the meter band is not executed. Multiple meter band can be defined, and the band is executed when the rate configuration is reached. This feature can be used to monitor packet entering the ingress port and apply metering or rate-limiting on their flows. In OpenFlow 1.4 [80], the controller is supported with a scheme to monitor a subset of the flow table(s) in the switch. This feature is added due to the support of distributed controllers, in which many controllers can manage the same network.

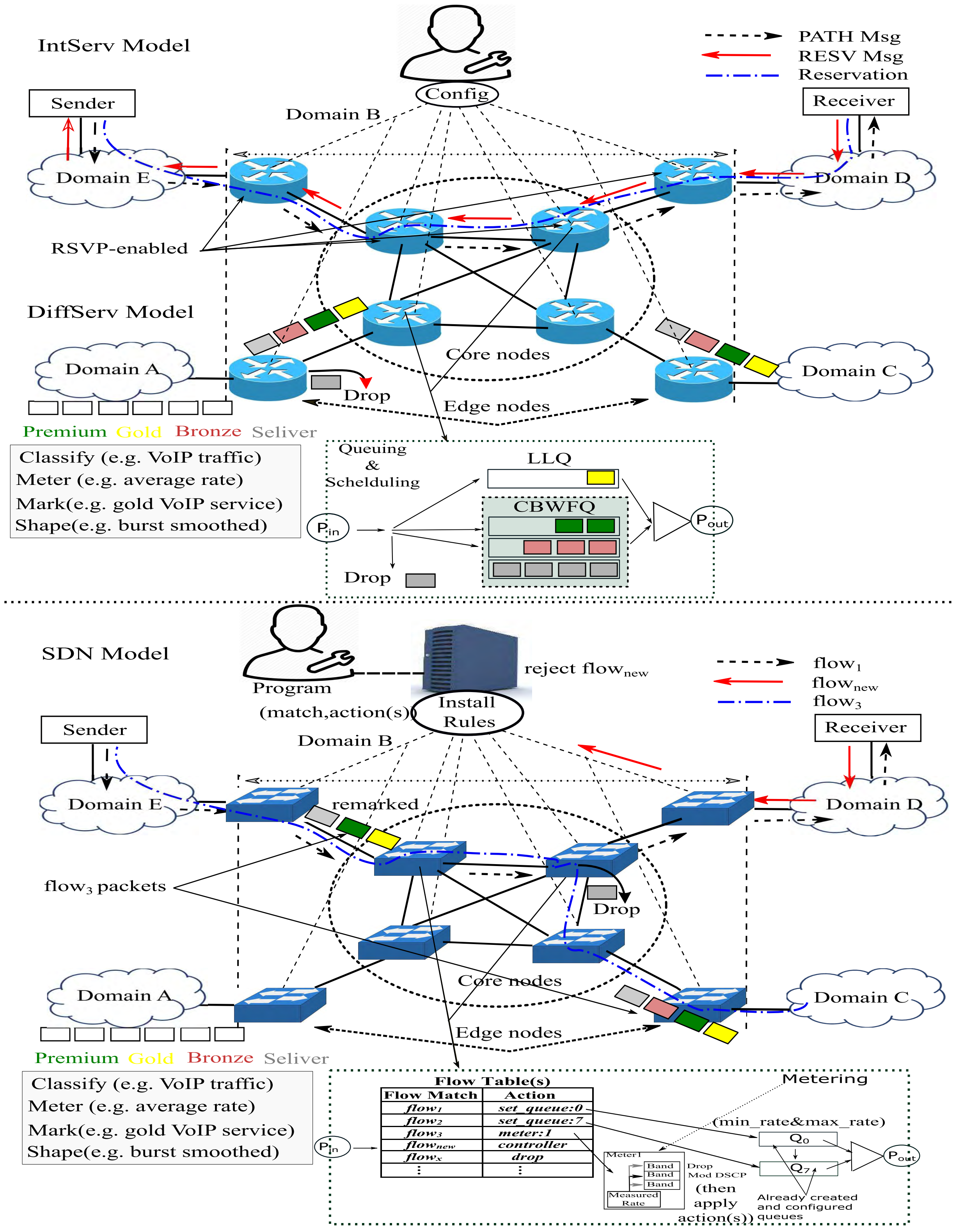


FIGURE 5. QoS in traditional and SDN networks.

Therefore, a controller can install monitors on a subset of the flow table, and whenever an addition, modification or removal or flows happen, an event packet is sent to the controller to inform it about that change. OpenFlow 1.5 [81] introduce new features such as the egress tables in output ports. It also adds idle time statistic to count the time this

TABLE 4. QoS in traditional and SDN networks.

	Traditional		**SDN**
	IntServ	**DiffServ**	
Granularity	Per Class	Per Flow	Supports Both
Admition Control	NA	At each node	At controller
Policing (drops)	At edge or intermediated nodes	Edge of the network and at all source merge points	At edge or intermediated nodes
Shaping	At edge nodes	At intermediate nodes	At edge nodes
Metering	Edge nodes	At edge of the network and at all source merge points	Configurable meters (i.e. meter table) at edge or intermediate nodes
Scheduling	At intermediate nodes, based on port configuration	NA	At intermediate nodes, based on port configuration, packets steered using set-queue action
Complexity	At edge nodes with long-term setup	At all nodes with per-flow setup	At controller with dynamic per-flow setup
Signaling	NA	RSVP protocol	OpenFlow protocol
State	Low, per-class at the intermediate nodes and per-aggregate at the edge nodes	Soft, need refreshing per-flow at each node	Configurable hard/time-based per-flow at each node
Provisioning time	Time consuming	Signaling time	Automatic configuration and short provisioning-time with fast rollback capability due to its programmability
Scalability	High	Low	Medium
Packet matching	Multi-Field	Defined at the reservation setup as *FilterSpec*	Multi-Field
Resource Utilization	Affected by routing	Affected by routing	Can be Optimized through dynamic central control of QoS and routing configuration
Service Scope	Domain	End-To-End	Domain

flow entry is idle. Also, TCP flags matching is added (i.e., such as SYN, ACK and FIN), in which it helps to detect the start/end of TCP connection. Moreover, flow statistics triggers are added to allow the switch to push-up statistics (i.e. beside the default poll-based) toward the controller when their configured threshold/timeout is reached. Furthermore, port configuration changes notification message is introduced to alert other controllers in a distributed deployment. Figure 5 and Table 4 compare implementation and main features of QoS in traditional and SDN networks.

IV. AUTONOMIC QoS WITHIN SDN

Autonomy frees up managers from network management duties and allows them to focus on improving business goals. With this rapid development of new services, network management will not be the same. Such services and quality demands are always changing which increases the CapEx/OpEx costs. A self-managed network is a dream of the higher administration level [10]. Autonomy is a suggested solution to overcome those challenges in deployment or management of such services. SDN can push toward achieving that goal with its generality and programmability which leads to ease deployment of new services. SDN network can be easily programmed to support the service or its users' requirements by writing a simple program module within or above the SDN controller with the conjunction of OpenFlow primitives. When it is over, the changes to the network can be rolled back by unplugging that software module. Therefore, it is a design choice to offer an autonomy of QoS management within SDN networks. However, in order to do it, such design should comply with those previously discussed functionalities of an Autonomic System.

Therefore, in this section, we organize the research effort in the literature of SDN-based QoS provisioning according to the MAPE-K model main functions, namely: *Monitor, Analysis, Plan, and Execute*. Then, we review and discuss the organization of the *Knowledge* source. We chose the MAPE-K because it is adopted by IBM for describing autonomic computing and it is the most famous reference control model for self-adaptive and autonomic systems. Also due to its clarity and modularity to describe how QoS autonomic provisioning and management should work. Therefore, we discuss each function as been tackled in the literature in the field of QoS management in SDN.

A. MONITOR

Monitoring of network services plays a key role. It helps to detect any degradation of the quality and ensures such services work correctly. Hence, problems can be resolved without interrupting such services or affecting the user satisfaction.

1) MONITORING IN OpenFlow

Each network system should know existing services and the state of its components to support QoS provisioning. SDN uses distance-based monitoring of network elements since the control is separated from those devices and located in a logically centralized place. OpenFlow provides within its specification a messaging mechanism that allows the controller to poll statistics (i.e., about flows, tables, ports, and queues) from those compatible devices [80].

In SDN, the monitoring function can be implemented within or above the controller to observe the network state. The controller internally keeps three different views of each device; capabilities, configurations, and statistics as shown in Fig. 6. Upon establishing the connection with the controller, each network device sends its capabilities (i.e., actually the controller asks for it by sending a feature request

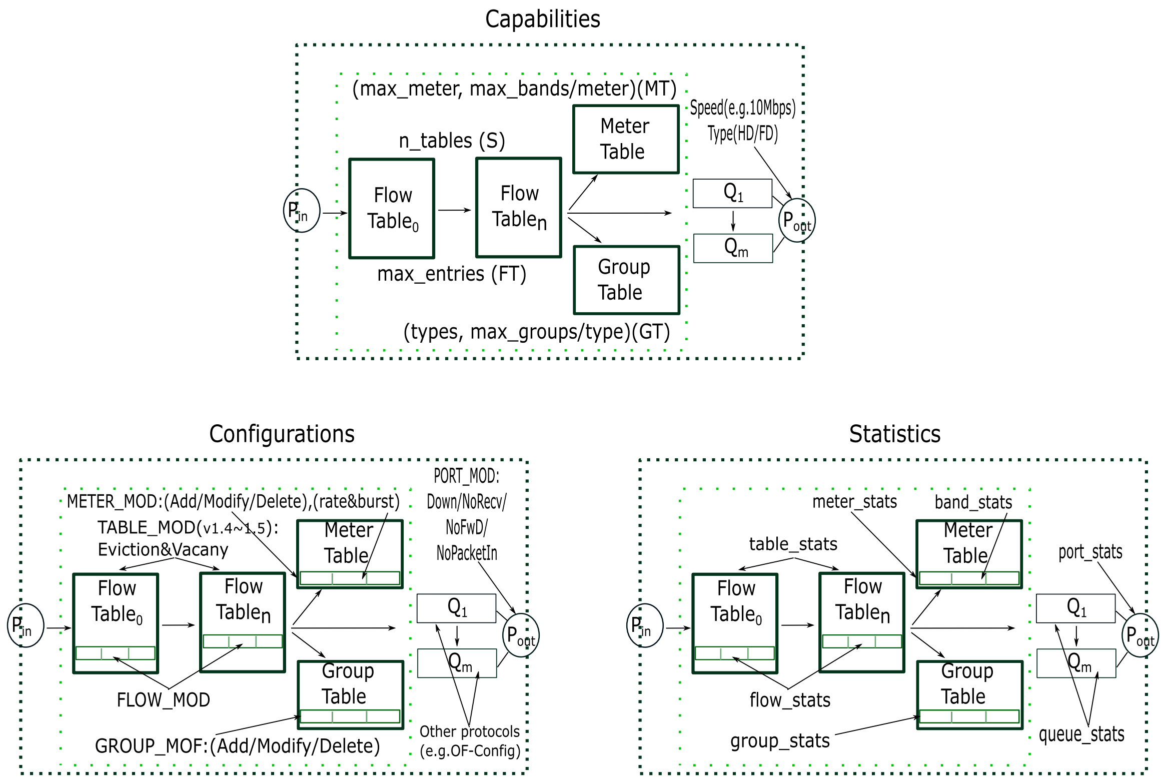

Capabilities
(max_meter, max_bands/meter)(MT)
n_tables (S)
Meter Table
Speed(e.g.10Mbps)
Type(HD/FD)
P_in
Flow Table_0
Flow Table_n
Q_1
Q_m
P_out
max_entries (FT)
Group Table
(types, max_groups/type)(GT)
Configurations
PORT_MOD: Down/NoRecv/ NoFwD/ NoPacketIn
METER_MOD:(Add/Modify/Delete),(rate&burst)
TABLE_MOD(v1.4~1.5): Eviction&Vacany
FLOW_MOD
GROUP_MOF:(Add/Modify/Delete)
Other protocols (e.g.OF-Config)
Statistics
meter_stats
band_stats
table_stats
port_stats
flow_stats
group_stats
queue_stats

FIGURE 6. Three views of OpenFlow (OF) Switch from the controller point of view.

to the switch). Features such as the maximum number of flow tables, and if the switch supports metering or grouping of flows and how many flow entries each table can hold. Moreover, it knows the speed, and the configuration of connected links and the number of queues each line has as shown in Fig. 6. This helps the controller to know what the network can do. Beside capabilities, the controller can get the current configuration of each device. To know the state of the network, the controller can send *StatReq* messages to poll statistics from those network devices as shown in Fig. 7.

Fig. 7 depicts the format of the control messages between a switch and the controller. Whenever a new flow arrives, a *PacketIn* message is sent to the controller informing it about this unrecognized flow packets. Upon reception of that packet, the controller replies by *FlowMod* message(s) to create a new entry for that flow within switches along the specified flow path. A flow entry holds counters for packets matching that flow. Moreover, other counters for that flow can coexist in other tables in that switch such as the meter table that is used to band the traffic generated by that flow. An entry expires after certain time of inactivity called *Idle-Timeout* or after a *HardTimout* elapsed even if the flow still active. At which, the switch removes expired flow entry and informs the controller by sending a *FlowRemoved* message (i.e., a flow-removed flag is used along with that flow entry). *FlowRemoved* packet contains statistics about that dead flow. Fig. 7 shows the structure of those counters in each switch and which OpenFlow version support. This approach used in SDN simplified the monitoring function. Simple counters are used and southbound APIs made available by the switch to allow the controller acquiring them. The authors in [84] investigated the OpenFlow monitoring capabilities and visualize the collected statistics in a GUI for the network operator.

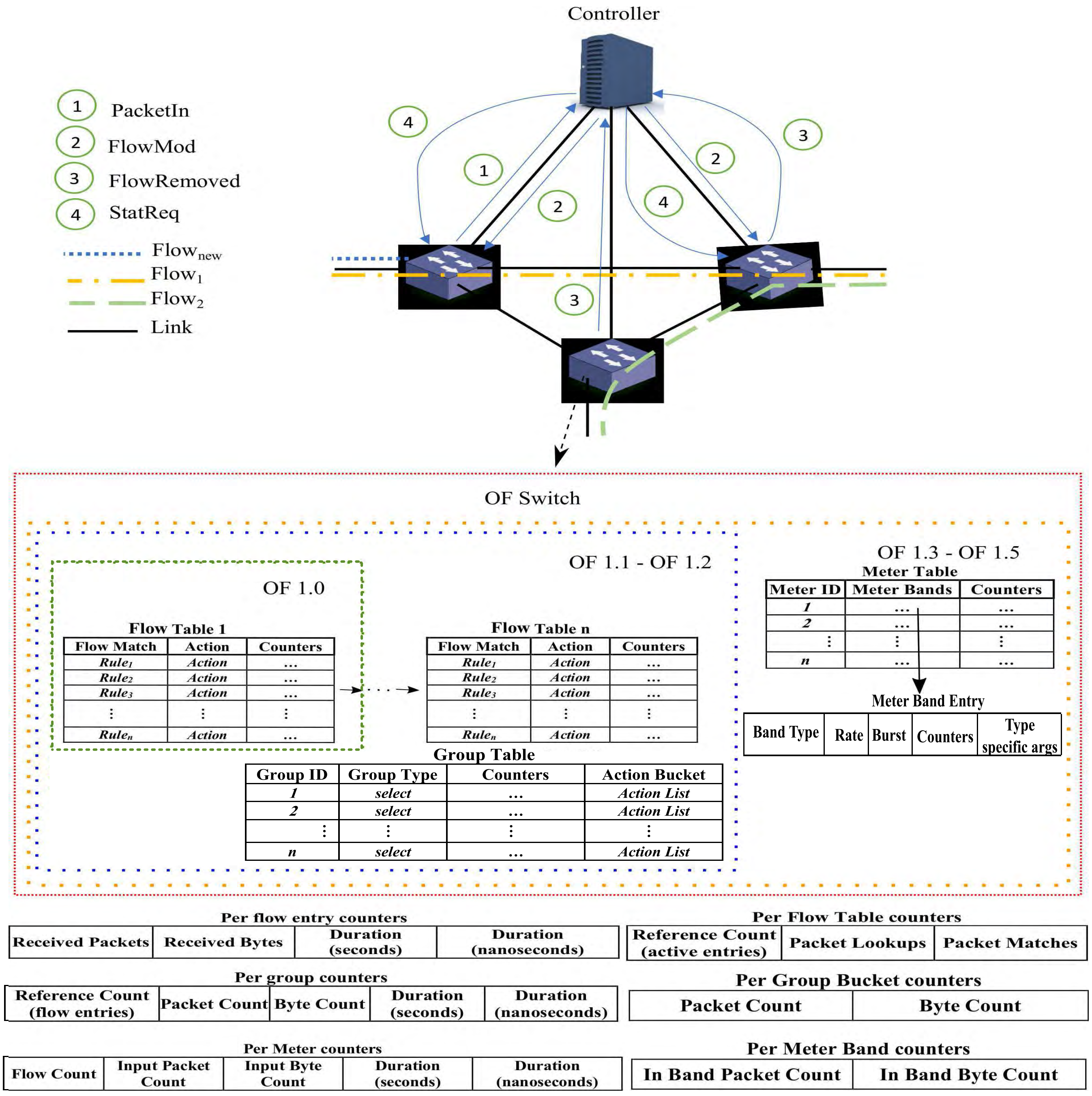
Monitoring is a key component in QoS provisioning. In order to comply with SLA requirements, accurate and timely measurements should be provided and made available [85]. In this section, we review and discuss research works related to network monitoring within SDN.

2) ACTIVE VS PASSIVE

The process of collecting measurements can either be active or passive. In the active measurement, initiation of that process usually done by the controller, not the switches. As mentioned before, controller sends *StatReq* messages to grab counters data stored within switches.

Most of the research work in [86]–[92], [94]–[100], [102]–[104], [106], [107], [109], [110], [112]–[116], [181]–[183], [195] uses the same approach.

The approach used in [86] collects network statistics through OpenFlow messages. It measures QoS metrics (e.g., utilization of port bandwidth and loss) and makes it available for upper applications such as traffic engineering. Those messages are sent periodically within a predefined polling interval (i.e., every 5 seconds). For instance in *OpenTM* [87], a single query is issued during each polling interval (i.e. also 5 seconds) for every different source-destination pairs. They



Flow Table 1

Flow Match	Action	Counters
Rule$_1$	*Action*	...
Rule$_2$	*Action*	...
Rule$_3$	*Action*	...
⋮	⋮	⋮
Rule$_n$	*Action*	...

Flow Table n

Flow Match	Action	Counters
Rule$_1$	*Action*	...
Rule$_2$	*Action*	...
Rule$_3$	*Action*	...
⋮	⋮	⋮
Rule$_n$	*Action*	...

Group Table

Group ID	Group Type	Counters	Action Bucket
1	*select*	...	*Action List*
2	*select*	...	*Action List*
⋮	⋮	⋮	⋮
n	*select*	...	*Action List*

Meter Table

Meter ID	Meter Bands	Counters
1	...	...
2	...	...
⋮	⋮	⋮
n	...	...

Band Type	Rate	Burst	Counters	Type specific args

Per flow entry counters

Received Packets	Received Bytes	Duration (seconds)	Duration (nanoseconds)

Per Flow Table counters

Reference Count (active entries)	Packet Lookups	Packet Matches

Per group counters

Reference Count (flow entries)	Packet Count	Byte Count	Duration (seconds)	Duration (nanoseconds)

Per Group Bucket counters

Packet Count	Byte Count

Per Meter counters

Flow Count	Input Packet Count	Input Byte Count	Duration (seconds)	Duration (nanoseconds)

Per Meter Band counters

In Band Packet Count	In Band Byte Count

FIGURE 7. ONF architecture for OpenFlow.

used source-destination pairs to identify flows and compute the volume of traffic in between. In [88], port statistics are periodically polled from network switches (i.e., every 500ms for accurate results) by the controller. Transmitted bytes by each switch's port is used to compute the consumed bandwidth between any two neighbor switches. In [89], controller polls network switches periodically (i.e., every second) to collects queues statistics. They used these retrieved statistics to compute queue's delay and the available bandwidth. Also, the authors in [182] periodically collect statistics to compute packets loss and estimates their effect on VoIP quality.

The authors in [83] proposed *FlowVisa* as a plugin module in the control plane. It is used to identify traffic flows that belong to the critical application. So those can be explicitly monitored or carefully. Instead of packet sampling or using deep packet inspection to identify that flow belong to which business application, *FlowVisa* communicate with such business application through northbound API to retrieve information that helps in digging the network traffic for those flows. By the identification of flows, switches that serve them identified and polled to receive statistic for those critical monitor flows. Using the same application interaction,

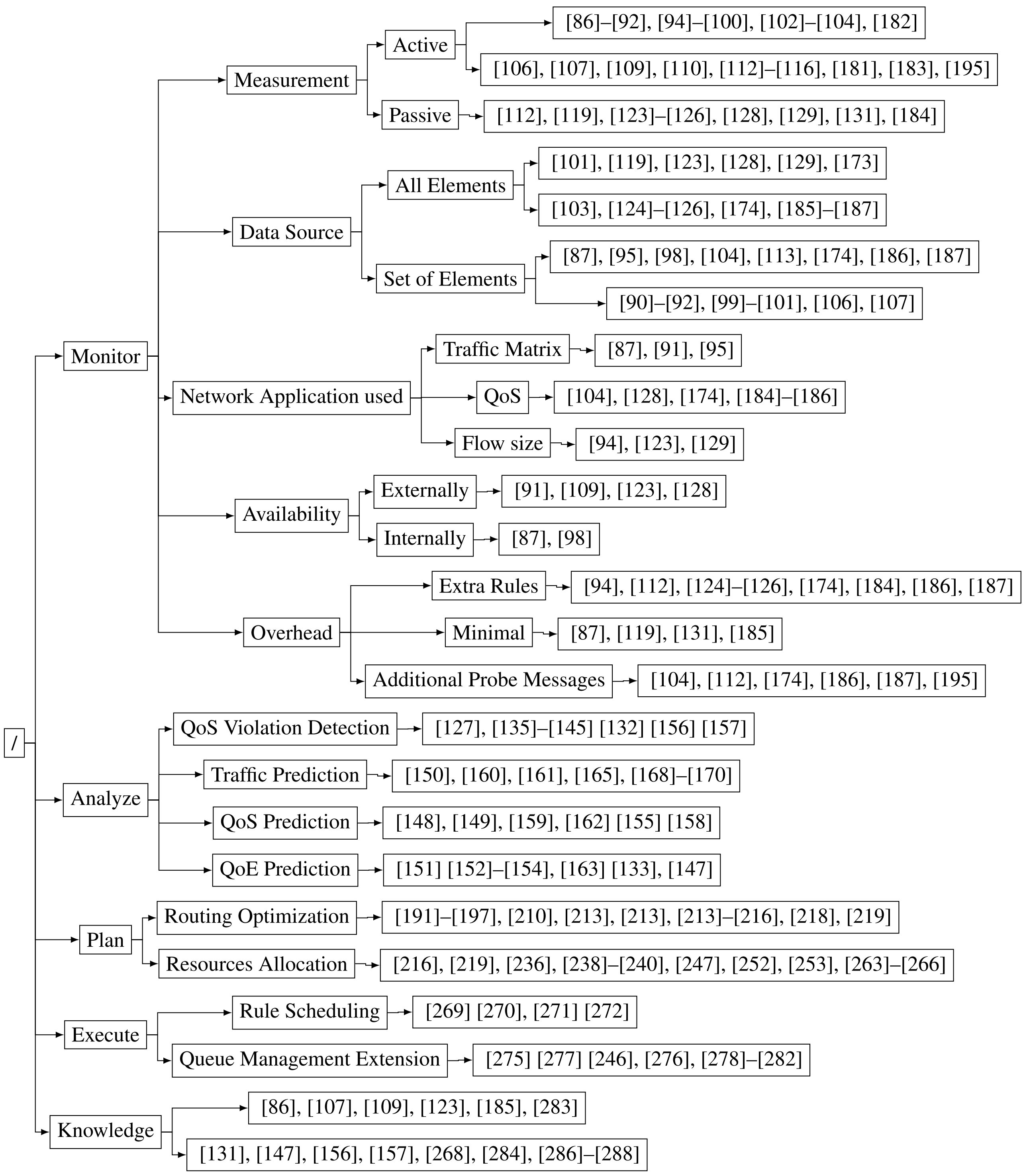


FIGURE 8. **Classification and taxonomy of prior work on SDN based on MAPE-K reference model.**

FlowVisa can get information about flow's lifetime which can help in stopping monitoring process before they expired in the flow tables as it will reduce the communication cost caused by *StatReq* messages.

Frequently used *StatReq* messages will overload the network. Therefore, some research works as it the case in [94] sacrificed accuracy for low overhead. They used IP prefixes as a matching rule for flows forwarding. By that, fewer per-flow statistics messages are sent along the network. Also, wild-card flow entries aggregation is an excellent choice to reduce statistic collection communication overhead. However, it loses fine-granular forwarding that many network applications are pursuing. Therefore, the authors in [90], [91] used an approach (i.e., called *CeMoC*) that achieves flows statistic aggregated collection while keeping fine-grained forwarding. Authors use the group table within the switch for that task. They create a group entry for each aggregated set of flows, with a group type set to SELECT. Each of these flows' action within the flow table points to that group entry identifier. Each group entry collects statistics for packets processed by that entry similar to those used in the per-flow within flow table (i.e., flows statistics aggregation). Each group entry contains a list of action buckets that is applied to those processed packets. Selection of which bucket to be executed depends on the criteria implemented by the SELECT group type. For instance, a hash value could be computed for the packet by the switch to choose between those buckets. Bucket action could be to forward the packet to a specific port similarly to those specified in the flow table entries. By doing that, the approach preserves the fine-granular forwarding. Instead of using per-flow (i.e., two messages) or poll-all (i.e., 1+ multi-part reply messages) statistic polling, the controller can polls group's statistic held within group entries. This approach may cause extra computation on the switch, and the selection criteria need to be carefully designed to achieve perfect forwarding. Fig. 9 shows the design of both works. It consists of a *RequestDispatcher* that receive user's statistic requests and converts it to low-level queries, a *FlowTracker* that tracks all active flows within the network, a *GroupMaker* that is responsible for creating group entries for the aggregated or grouped flows, a *QueryMaker* that polls statistics from the selected switches by sending *StatReq* messages and a *Collector* that collects statistics received from network switches. In [90], a *PollingScheduler* module configures the polling interval. In [91], that module shifted to the upper coordination layer.

Besides the communication cost trade-off, TCAM should be wisely used when installing flows entries either for forwarding or monitoring [93]. In *OpenMeaure* [92], the authors start by installing destination-based coarse rules for routing. Then, an initial flow-size estimation is obtained from those periodically collected statistics. Online learning is used to identify the most interesting flows to monitor. Therefore, fine-grained rules are installed which allows *OpenMeasure* to select interesting large flows to track and measure.

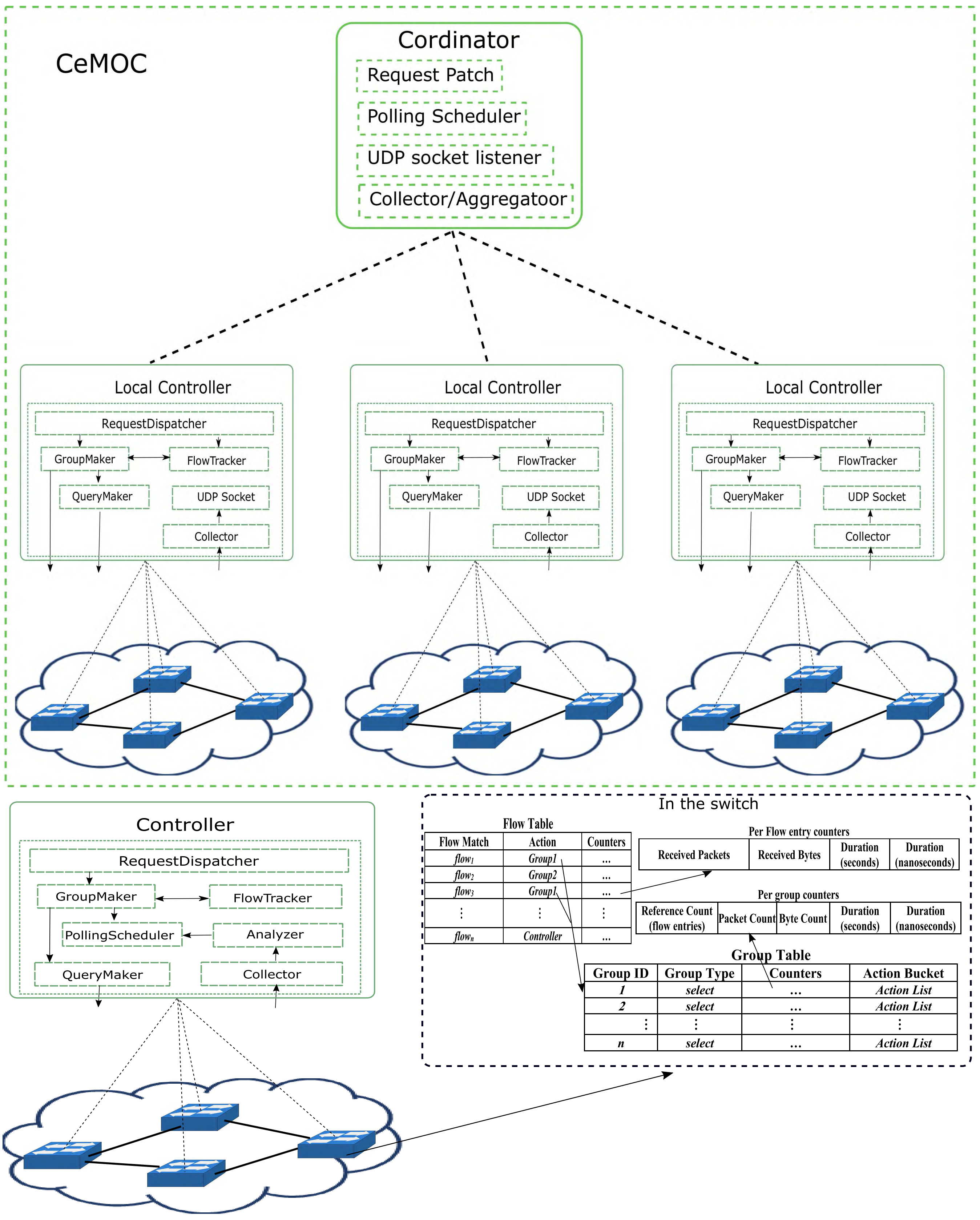
The authors in [94] exploit the way OpenFlow forwarding works to monitor the network. They use a set of rules within the switch to match traffic that under monitoring. Moreover, by changing rule priority, the controller can dynamically changes which the most critical traffic to monitor. The approach they used is simple, and their objective is to overcome the need to change data structure for monitoring purpose. They scarify the accuracy with low overhead since large aggregation is performed in rule installation (i.e., they used IP prefixes as matching rules). They adapt those rules to focus on flows that are more likely to have high traffic volume, i.e., to tackle the problem of hierarchical heavy hitter detection). They assume that there is only monitoring rules and no actions that drop or forward the packet which not the case in current OpenFlow networks.

The authors in [176] focus on getting fine-granular measurements of more rewarding flows. From their perspective, most massive volume flows are the most important candidates for per-flow monitoring. Therefore, they used an intelligent Multi-Armed Bandit (MAB) algorithm to select those flows and stamped as important. Part of flow entries in switch TCAM are reserved to fine-granularly measure them while the rest is flows are optimally aggregated, and flow rules are installed at the other part of TCAM. This division is evaluated dynamically within a specific monitoring interval.

The authors in [177] decouple monitoring from the forwarding process. They create a separate monitoring table with match fields similar to those in the flow table. However, the decision of monitoring of flows is chosen by the application through the controller by changing monitoring match fields in the monitoring table. This process is controlled in the switch by the local control application that uses configured bloom-filer entries for faster monitoring decision. For instance, the incoming packet head matched the bloom-filter element, and this packet is not monitored. Otherwise, the packet header is matched with those entries in the monitoring table. Counters are updates if there is a match. Otherwise, a sampling ratio is used to decide to monitor this new flow or not. If the decision was not to monitor this flow, a new element is created in the bloom-filter to announce non-monitoring for this flow in the future. These statistics are stored in a database within the switch which polled by the controller based on the applications' requirement. This work requires a change to the switch and may add more processing, and storage overhead to it. The controller glues between the application and the switches through the monitoring APIs. They are used to configure the monitoring process within the switch (e.g., enable or disable, change matching fields). Then the authors tried in [178] to distribute the monitoring load of flows between network switches. Since the controller has the global view of the monitoring data, flows that are monitored by multiples switches are filtered. Switches with the highest number of monitoring entries remove the monitoring entry belong to that flow from their tables. Moreover, in [179], they considered switch resources. They choose switches that have enough memory or computation (i.e., idle CPU time).

The authors in [96] periodically collect port and flow statistics for packet loss calculation. However, and differently from previous works, the authors pointed that non-data traffic is contributing to such statistics (i.e., port specifically). Therefore, and for accurate loss estimation, non-data statistics need to be discounted from collected measurements based on their previous work in [97] that estimates non-data traffic in the network.

The authors in [95] proposed schemes to reduce the communication cost imposed by request/reply messages, for global monitoring of SDN network, by producing optimized polling schemes for network switches. Instead of polling all switches along the path of a single flow, they heuristically choose switches that cover most of those active flows. This process continues until all active flows are covered.



Flow Table

Flow Match	Action	Counters
flow$_1$	*Group1*	...
flow$_2$	*Group2*	...
flow$_3$	*Group1*	...
⋮	⋮	⋮
flow$_n$	*Controller*	...

Per Flow entry counters

Received Packets	Received Bytes	Duration (seconds)	Duration (nanoseconds)

Per group counters

Reference Count (flow entries)	Packet Count	Byte Count	Duration (seconds)	Duration (nanoseconds)

Group Table

Group ID	Group Type	Counters	Action Bucket
1	*select*	...	*Action List*
2	*select*	...	*Action List*
⋮	⋮	⋮	⋮
n	*select*	...	*Action List*

FIGURE 9. Illustrative diagram of CeMOC as reported in [90], [91].

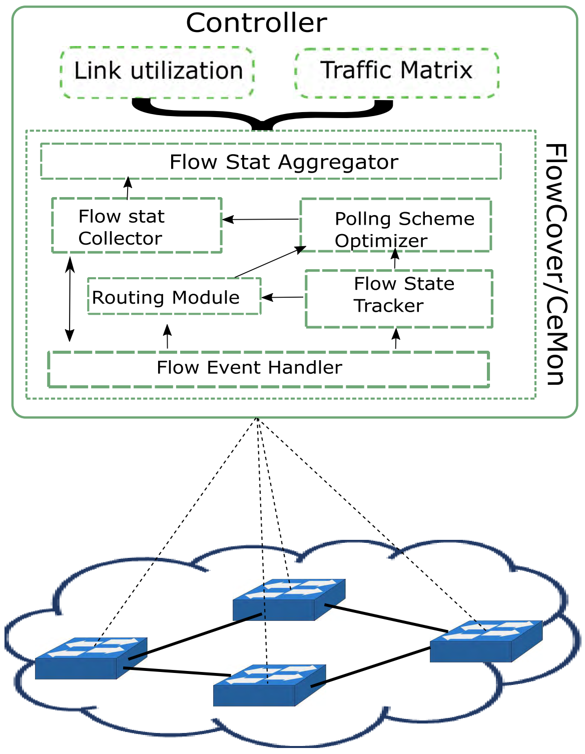


FIGURE 10. Illustrative diagram for FlowCover/CeMon scheme as reported in [95], [98].

Fig. 10 shows their proposed approach. It consists of five modules: *flow event handler* which handles flow arrival or expiring messages and forward it to *flow state tracker*. It holds and maintains a list of active flows within the network. Then a *polling scheme optimizer* uses that list of flows and their routing paths (i.e., from the routing module) to produce an optimized polling scheme. It is used by *flow stat collector* to poll statistics from switches. Received data are passed to *flow stat aggregator* to make it available for upper applications. If a new flow is detected by the *flow state tracker*, the optimizer checks if it been covered by an already generated scheme, if not a new polling rule adding to the existing scheme. Polling scheme is recomputed periodically by the optimizer to achieve the most up to date view. They assumed out-band control in which the distance between the controller and switches is assumed to be the same. However, in [98] they added in-band control network in which the location of the controller is counted (i.e., or the hops to the switch is counted) onto the cost of polling the switch.

In [99], [100], the authors proposed a method to minimize the number of polling queries by limiting it to switches that offer valuable information to the monitoring process. Therefore, they cluster the switches based on the number of ports and their criticality. The switch criticality is defined as how many times it appears in the forwarding paths from a client to server hosts. Switches with three or more ports are cluster based on their criticality. A polling algorithm randomly chooses switches from such a cluster. By doing that, not all switches are queried together in the same monitoring interval t_{mon}.

In [102], the authors propose a greedy polling algorithm called lonely flow first (LFF) in which the lonely flows are covered first with a poll-all message. Lonely flow is a flow that passes through only one switch. By finding such flows, the algorithm divides the network switches into two groups, with/without lonely flows. They define two communication costs that are taken into account, the distance and the message overhead (i.e., request/reply messages) for using poll-single or poll-all. Within each group of switches, the cost is compared, if the poll-all is higher, the poll-single messages are used and vice versa until all flows are covered by which the polling cycle finish.

In [90], the authors design an adaptive polling interval algorithm that reacts to traffic fluctuation. It uses the difference between the current and the previous traffic measurements for a group of flows to detect the fluctuation. A shorter interval is used when the fluctuation is high and a longer interval otherwise. Also, the authors in [95], [98] use an adaptive approach to compute the polling frequency of sampling interval. It adjusts the frequency according to traffic changes, hugely changing traffic requires shorter polling period. In order to alleviate fluctuation, they used an adaptive sliding window of traffic history for smoothing the changes.

In [103], the authors poll flow's statistics from a network switch according to a predefined polling interval (i.e. min(0.5s), max(5s)). However, to detect the fluctuation in flow's rate, the static configuration is not enough and adaption is required. They use a sliding window queue (i.e., queue size = 3) to keep the history of consequence byte counts measurements. Then, the difference between the current and last measurement (i.e., transmission within the last interval) is compared with this sliding window queue mean added to its standard deviation. If the new value is larger than the mean and standard deviation sum, the polling interval for that flow is decreased, and its sliding window queue size is reduced also. Otherwise, the polling interval and the queue size is increased. That adaptive approach is called *AdaRate*. Then, they made another modification in which a random choice is made to adapt according to *AdaRate* or to use a minimum polling interval (i.e., called *RAdaRate*) to detect any missed fluctuation between two measurements.

The authors in [105] adapt the polling frequency according to ports/switches activity. They use the exponential moving average to reduce the effects of instantaneous fluctuations while maintaining measurement accuracy. In *OpenNetMon* [104], the arrival of new flows is used in adapting the polling interval as an indication of the network the activity. Thus, switches are polled at a faster rate. Otherwise, when the state is static or no changes in flows routes, counters are polled periodically.

Based on *OpenNetMon*, the authors in [106] built their architecture and called it *OpenLL* (OpenFlow-based Low-cost and Low-error). They use adaptive sampling in order to reduce the communication overhead and improve accuracy. They compared the difference between the current and previously computed flow throughput. Also, adaption coefficients are to change the polling interval. The interval is limited by a low (i.e., 1s) and a max(i.e., 30s) bounds. The authors in [107] used a similar approach to measure a flow rate. Within each polling interval, N requests are sent to both first and last switches (i.e., each switch receive $N/2$ requests unless N is not an even, then the last switch is polled once more than the first one). The polling interval is adapted and bounded between two values, a min(1s) and a max(15s). Authors use historical computed flow rate values to predict future ones. Then, they compute the change of the prediction based on the changed value. The interval either increases or decreased or stay the same. For instance, if the changes are faster than recent prediction, the polling interval is decreased. They used the same concept used in TCP congestion control to change the value of the polling interval. An additive increase (i.e., incremented by 1) and Multiplicative decrease (i.e., divided by 2).

For active flows identification, the authors in [109] monitored the arrival of *PacketIn* and *FlowRemoved* messages. Even this approach is not practical in the proactive flow entries deployment. For polling schedule, they used a time-out (t in milliseconds) for both the flow to expire and polling request to be issued. If the flows *FlowRemoved* message is received within that interval, the statistics will be embedded within. Otherwise, the flow statistics are polled. The difference between the counted value and the last measurement is computed and compared to a specific threshold. If the difference is lower, then the time-out value is increased by a multiple of a constant value. If the flow rate still slow, the process repeated until it reaches some maximum value. This process is performed similarly if the flow rate is high by which the time-out value is decreased by dividing it with another constant until the maximum value is reached. This process may cause overhead. Therefore, initial time-out values need to be carefully chosen.

The authors in [110] proposed a model to capture the nature of the behavior of a flow's traffic or volume by sampling flow's packets. They implement two sampling algorithms; a stochastic and a systematic. The first one samples flow's packets based on certain probability while the other uses sampling ratio (i.e., sample m packets after N consequence ones) similar to sFlow. Based on these sampled statistics the polling interval is adapted. For instance, if the traffic is stable, the scheduler increases the interval. Otherwise, it uses a shorter one (i.e., 0.5s to 5s is the used interval bounds). So, it adapts to have higher accuracy when necessary and reduces the overhead when it is not worthing like the work in [107]. Therefore, the flow sampling is used to help the controller adapt flow's polling interval. Implementation of the sampling algorithms requires changes to the data-plane.

Differently that previously discussed works, other types of measurement cannot be inferred from polled switch counters. For instance, the authors in [195] proposed a framework called *SLAM* to monitor path latency in SDN-based data center networks. High delay network segments need to be detected. Authors used a customized probe packet to trigger *PacktIn* notification packets. Then it measures the latency based on the arrival time of the *PacktIn* message to the controller. Also, the authors, in [113] used special packets injected to test end-to-end delay for flows. Each packet travels the path from first to the last switch and goes back to the controller. This process is repeated periodically for each monitored flow. And similarly, the work in [112], [183] and in *OpenNetMon* [104].

In [114], probe packets are injected to test link delay. Instead of using static short paths and due to computation expensive path optimization, they used a random walk based on weighted links that satisfy a min-max probe rate for each link. The authors in [115] utilize the same probe-based approach to obtain the link delay. Instead of probing for every link, in [116], the authors used the network graph built by the topology discovery module to produce a tree that covers all links with the least number of hops. Then, sub-paths are probed similarly to previous works to path delays. Therefore, to compute the delay of a link in a specific tree level, its previous levels links delays are subtracted from its sup-path delay. So, there is no need to probe switches for link delay if that link resides in a sub-path that its delay is computed.

The authors in [180] used an approach called Bidirectional forwarding detection (BFD) that detect link failures in OVSs. In which switches send messages in between, absences of such message means link failure. Authors implemented an echo mode in which a switch sends echo messages between each other through those connected ports. Then, by computing the difference of reception and sending times, link delay can be estimated. This value is stored and added to LLDP packets when passed by the switch to be forwarded to the controller.

The authors in [181] use test machines to send probe packets periodically that travel along structured rings. Those probes are used to test link delay and loss ratios.

Active measurement provides accurate results; however, it leads to more overhead due to request/replay statistic messages or probe packets. The authors in [109], [110], [117]–[120], [123]–[129] proposed ways to reduce such overhead by pushing up statistic toward the controller.

In [117], [118], flow discovery is used to identify which flows need to be monitored, the proposed approach called *NFO*. For this purpose, *NFO* installs aggregated flow entries called discovery entries within a set of network switches in a way that balances the monitoring load. Available capacity of the flow table is used to direct the discovery module in which switch(s) those entries are installed. The switch sends the matched packet to the controller (i.e., action "to controller" is used in the installed discovery entries). Upon reception of *PacketIn* that been generated due to matched discovery flow entry, the controller installs an exact match entry for that identified active flow. A push-based approach is implemented by setting a timeout and flow-remove flag along with this newly created flow entry. The timeout value determines the frequency in which statistic is pushed for that flow. Every time a flow expires, a *FlowRemoved* packet is sent to the controller with counted statistics. In response to *FlowRemoved* message, a flow entry is created by the controller (i.e., if the flow needs to be monitored) with a new timeout value computed by a frequency scheduler module similar to that used in [109].

In [119], the authors leverage OpenFlow control messages to monitor network utilization passively. In OpenFlow, a *PacketIn* and *FlowRemoved* packets is sent to inform the controller the new and expired flows respectively. Authors used those two messages to create discrete time checkpoints depict the lifetime interval of each flow. Along with the *FlowRemoved* packet, a statistics about count bytes matched and duration of that flow.

The authors in [120] proposed a solution to reduce communication cost by reducing the rate at which statistics are polled using *StatReq* messages. They leveraged a new feature in OpenFlow 1.5 that allows the switch to push flow statistics based on a preconfigured threshold or its multiples. They also utilized the statistics contained within *FlowRemoved* messages to enhance the granularity with no cost. If both previously discussed method was not enough in the polling interval T timeout, a *StatReq* message is sent to poll statistics for that flow. The approach uses the rate in which it receives the flow-stat trigger messages to adapt the threshold, and hence reduces the overhead. It increases the threshold if the trigger rabidly ignited and vice versa to increase the flow-stat triggering by reducing that configured threshold.

Instead of using OpenFlow messages, the authors in [123] used sFlow protocol [121] as their source of the network measurements. sFlow is a packet sample protocol that takes one sample from N packets that traverse the switch and sends packet's header with some meta-data toward a central sFlow collector which is the SDN controller in that work. Therefore, the accuracy depends on N and the rate is variable due to N and the variability of packets arrival rate. Authors used sFlow due it is stateless as it is not the case in NetFlow [122] since the sampled packet is sent automatically to the collector.

In [110] authors implement two per-flow sampling algorithms; stochastic and systematic. The first one samples flow's packet based on a certain probability while the other uses sampling ratio (i.e., sample m packet after N consequence packets) similar to sFlow [121]. They used these sampled packets to tune polling interval by the controller. Similarly, the authors in [111] modified actions specified with flows' entry to implement sampling. An m out of n consequence packets (i.e., deterministic) or with probability p (i.e., stochastic) are sampled from the monitored flow.

The authors in [124]–[126] implement a NetFlow [122] similar sampling using OpenFlow existing features. They developed two methods: one is an IP prefix based in which controller install special monitoring rules that match certain IP prefixes by matching the last n bits of the source and destination IP address. They claim that the sampling rate depends on the number of bits masked. The other method computes a hash value of the 5-tuple of the packet header and checks if this computed value is within a particular range. In both methods, the switch sends the matched packets for those monitored flows to the controller. This approach requires the installation of a set of monitoring rules within the OpenFlow switch which may consume more spaces if it needs to monitor more granular flows. In [127], the switch monitors a link statistics and send its statistic periodically toward the controller. Therefore, the controller gets a final value to compute a link utilization table.

Some OpenFlow enabled switches may also support other measurement protocols such as sFlow. The authors in [128] exploited such feature and developed a monitoring module for the Floodlight controller called *SDNMon*. It used two approaches for collecting statistics, polling through OpenFlow statistic messages and sampling through sFlow protocol. Similarly, the authors of [129] used two random sampling, per-packet and per-byte. The switch send those sampled packets to the controller for heavy flow detection modules that uses them to detected suspicious massive flows based on some thresholds. Upon suspicion, count rules are installed within that switch to count the flow. These counters are polled periodically by the controller for the heavy flow detection module.

3) FROM PART VS ALL ELEMENTS

The source of measurements has a primary effect on the monitoring process. There are two directions in the literature: polling part or all of the network elements.

In *OpenTM* [87], the authors try different algorithms such as querying the last switch only, round-robin, non-uniform random and others. They found that polling the last switches before destination gives the best accuracy but adds more load on them. While in *OpenNetMon* [104], the authors decide to collect statistics from ingress and egress switches only. Moreover, instead of polling, all switches along the path of a single flow, authors in [95], [98] heuristically choose switches that cover most of the active flows. This process continues until all active flows are covered. Works done in [113], [174], [186], [187] make the two end switches. Where the path's delay is under test, send probe messages toward the controller. In [90], used approach polls group statistic from edge switches in a data center fat-tree topology. In [91], a coordination layer arranges the operation of flows measurements collection within a distributed control environment (i.e., multiple controllers coexist). At the coordinator, a switch selection process is performed with the purpose of maximizing flows' coverage. Then a controller selection process is performed to minimize the communication cost, propagation delay, and controller overhead.

In *OpenMeaure* [92], after interesting flows are identified, two heuristic algorithms are proposed to do rule placement within network switches. Therefore, from candidates, a switch is chosen to monitor the flow if it had enough TCAM entries and closer to the destination (i.e., last hop approach) or it has the maximum entries for covering a minimum number of flows (i.e., greedy approach). Differently, than in *OpenTM*, it uses the last switch only, the authors in [106] collected measurement from edge switches (i.e., first and last) that flow passes through and the same is done in [107].

In [99], [100], switches with two ports (i.e., In and Out) only labeled as a 2-grade cluster in which data rate going from them are inferred from their others neighbor switches. Therefore, there is no need to query them. Other switches, with three or more ports, are cluster based on their criticality. A polling algorithm randomly chooses switches from such clusters. By doing that, not all switches in a cluster are queried together in the same monitoring interval. This reservation works well only if such two port switches exist in the network which is not always the case.

The work of authors in [101] divides the monitoring of flows between switches in which 5-tuple fine-granular rules installed to monitor application level flow in a switch for a chosen pair of the hosts while other switches in the path use 2-tuple coarse rules for forwarding. Therefore, with the same number of rules and instead of using ingress switches, application flows can be monitored.

In [103] all switches are polled. Also, in [89] all switches by the controller are polled periodically (i.e., every second) to collects queues statistics. Similarly, all switches are polled in [86] to collects network statistics through OpenFlow messages. In [88], the authors use the topology that is maintained by the controller to query network switches. Transmitted bytes by each switch's port is collected and used to estimate link capacity. In [127], all switches are monitored in which port statistics are computed and pushed toward the controller to get a global macro view of the network utilization. In [96], port and flow statistics are polled from all networks switches. Similarly did authors in [155], [158], they periodically collect port statistic from all network switches.

Instead of acquiring statistics, in [123] authors used sFlow as the source of measurement. Sampled measurements are sent from network switches toward the controller. Of course, idle ones are not considered here since no flows packets to pass along. Similarly, the authors in [128] use sFlow along with OpenFlow to get statistics from all devices. Similarly, the authors in [124]–[126] implement a NetFlow [122] similar sampling using the OpenFlow exiting features. All switches along the path are expected to send matched packets toward the controller. Even the work in [119] uses only OpenFlow *PacketIn* and *FlowRemoved* messages to collect measurement, all active switches are expected to be a source of such data. The work has been done in [186], [187] made switches along the tested path send *PacketIn* messages toward the controller to compute links'delay.

4) OVERHEAD

Achieving accurate measurements is always drawn back by cost in memory and communications. In this section, we discuss reviewed work from that point of view.

a: MINIMAL OR BASIC

In [119], the authors leverage OpenFlow control messages to monitor network utilization passively. In OpenFlow, a *PacketIn* packet is sent to inform the controller about the unmatched packet, upon reception of it the controller sends flow installation message that matches this new flow packets and programs the switch how to forward them. Along each flow, an expiration period after which if there are no packets received from that flow, the flow is expired and removed from the flow entries. After that, a *FlowRemoved* packet is sent to inform the controller of the flow entry removal. Authors used those two messages to create discrete time checkpoints depict the lifetime interval of each flow with zero extra overhead. However, they scarify the accuracy. Authors of *OpenMeaure* [92] proposed schemes to adapt the monitoring process by obtaining a coarse-grained measurement at the beginning. Then additional fine-grained measurements are collected from most rewarding flows (i.e., large flows). In [96], the authors poll all network switches every k seconds. Two requests are sent within every round for every switch ($2N$ within every k seconds, where N is the number of switches). Work in [109] depend mainly on *FlowRemoved* packets for measurements collection. A flow is polled if a timeout counter is expired without receiving any information about that flow. Both [86], [87] poll the network every 5 seconds which adequate to get good accuracy.

b: EXTRA RULES

In [94], the authors use a set of monitoring rules with the switch to match traffic under test. A monitoring rule updates counters without forwarding actions. By changing rule's priority, the controller can dynamically change which most vital traffic to monitor. The approach they use is simple, and their objective is to overcome the need to change data structure for monitoring purpose, also by doing that they scarify accuracy with low overhead. Since high aggregation is performed in rule installation (i.e., they used IP prefixes as a matching rule). Also, the approach used in [124]–[126] requires the installation of a set of monitoring rules within the OpenFlow switches which may consume more spaces if more granular flows need to be monitored. They implement a NetFlow [122] similar sampling using OpenFlow exiting features.

All latency or link delay approaches that depend on in injecting probe packet or specially crafted packets to make switches send the *PacketIn* message toward the controller requires extra rules to be installed along paths followed by those packets as it is the case in works done in [112], [174], [186], [187]. In [114], *SDProber*, probe packets travel through the network after adding special forwarding rules for them. Those rules are modified to route the probe packets toward areas that congestion could occur. In [90], [91], the authors utilize group table to accumulate grouped flows statistics. Therefore, a new entry is created within the group table for every set of flows shares a specific switch link.

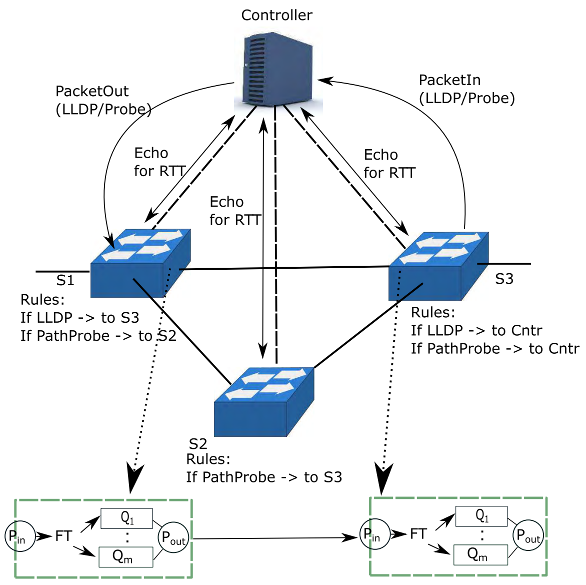


FIGURE 11. Link delay measurements in SDN.

c: ADDITIONAL PROBES OR MESSAGES

In all of the reviewed works, testing a link or path latency is done through injecting packets that loop through that path until the controller receives it. Fig. 11 shows how that process is done. The authors in [104], [112]–[114], [116], [174], [183], [186], [187], [195] use similar ways to compute link latency with minimal differences. In [104], the approach uses injected packets that travel between ingress&egress switches along the same interested flow's path and come back to the controller which computes the delay it took. In [112], the authors used probe-based approach to test the delay of a specific path. Controller injects a special packet that loop over the chosen path and at the completion of a loop, a copy is sent to the controller.

In [174] authors injected a packet into the network that loops along the tested path and goes back to the controller to compute the latency from its departure and arrival times. Similarly, did the authors in [183], the authors in [195] proposed a framework called *SLAM* to monitor path latency in SDN-based data center networks in order to detect high delay network segments. Authors use a customized probe packet to trigger *PacktInt* notification packets toward the controller when a new unmatched packet is received in the network switch. Then, it measures the latency based on the arrival time of the *PacktInt* message to the controller similar to the work been done in [104], [109]. Similarly, the authors, in [113] use special packets injected to test the end-to-end delay for a flow. Each packet travels the path from the first to the last switch and goes back to the controller. This process is repeated periodically for each monitored flow. Also, the approach been used in [175] requires probe packet to travel the link to estimate its propagation delay.

The authors in [186], [187] use *LLDP* packets instead of probes for link discovery and delay estimation. Echo messages are used to tell the delay between the controller and switches.

The authors in [106] injected probe packets that travel the path between the first and the last switches and goes back to the controller to estimate the per-flow delay. Similarly did the authors in [114]. However, they tried to control the rate by which links are probed within defined bounds defined by the network operator which leads to resource conservation and to detect delayed links in a timely way. Also, there is a maximum threshold for probes per a minute for the whole network. By using those constraints, optimization of route selection will consume computation. Therefore, probes' routing is done based on a random walk and the probe rates. They used a weighted network graph that is reflected on the network's switches and its connecting links. High weights mean higher probability that the node and specifically that link will be probed in the next route. Dynamic changes of those weights allow the adaption of the probe process to comply with the probe rate constraints.

In [89], the authors utilized the same probe-based approach. However, they test the link's queue delay. Similarly, the authors in [115] utilize the same probe-based approach (i.e. similar to LLDP packets in Fig. 11) to obtain the link delay. However, they optimize the measurement process to reduce the overhead (i.e., on controller and network). Therefore, the time of the next link delay measurement is chosen based on a sequence of the previous measurements. The jitter of a sequence of N, recent delay, measurements is analyzed to determine the detection interval. Accurate control of it keeps results accurate and reduces the overhead. The sequence of measurements is updated after each measurement and continuously analyzed to detect the delay changes that will affect the next measurement time.

5) AVAILABILITY

Collected measurements would not be useful unless other modules analyze it. Therefore, availability in a natural way is an essential advantage for QoS provisioning.

a: INTERNALLY

Work done in *OpenTM* [87] was intended to compute volume of network traffic. Collected measurement are used to builds traffic matrix. It has been made it available internally for other controller applications. Similarly, per-flow and per-port statistics are collected by the approach in [128] stored and made available for internal access only. Other modules such as multi-path routing module can get the benefit of it. Similarly did the authors in [98].

b: EXTERNALLY

It is not necessary for QoS application or any other application plane modules to be implemented within the controller. Therefore, the accessibility of information offered by the controller is a necessity especially monitoring measurements. The authors in [109] implemented a monitoring framework on top of Floodlight controller to offer statistics polling service for network applications. They provide RESTful API from which application can access those collected statistics about network state with different aggregation levels. Similarly, the authors in [91] tackle the problem of flow measurement collection in a distributed controller environment. The architecture consist of a set of controllers each manage a set of switches or flows within different switches and an upper ordination layer to collect and aggregate all measurements received from those different controllers. Each controller makes its collected measurements available for the coordinator through UDP datagram socket.

The approach used in *OpenSample* [123] collects data from the network and build a snapshot of it to offer elephant flow detection and link utilization information. The produced network snapshot is offered and made accessible through an API for application such as traffic engineering as the authors used for their evaluation. The authors in [86] developed a network monitor application called OOFMonitor. That informs Ryu controller through northbound API to collect network statistics through OpenFlow messages. In that design, *Statistics Manager* is the module within the controller that receive requests from the application and offer collected statistics through JSON API. Similarly, the authors in [82] implement a network monitoring application called *SOFTmon* that leverage northbound API (i.e., REST) to interact with the controller for statistic collection process. Then collected data are visualized for the network operator.

The approach used in [86] collects network statistics through OpenFlow messages. Then it computes four (utilization of port bandwidth, delay, jettier and loss) QoS metrics and makes it available for other applications such as traffic engineering or QoS routing through JSON northbound APIs.

6) APPLICATIONS

Many network applications can benefit from collected measurements by the monitoring function. However, we focus on applications more related to QoS provisioning as follows.

a: TRAFFIC MATRIX

The authors in [87] leverage flow tracking within switches (i.e., counter per flow) and routing information within the controller to build a traffic matrix estimator. Traffic matrix usually helps in load balancing applications since it represents the traffic amount between all possible pairs within the network. OpenMeasure [92] used online learning to estimate per-flow size and identify expected large flows. By adapting rule installation, a list of flows and there sizes that are used to estimate Traffic Matrix. Furthermore, the authors in [90], [91] used group-based flow statistic aggregation in network switches to collected measurements and use it to estimate traffic matrix in a data-center environment. Similarly, the authors in [95] utilized monitoring measurements to estimate traffic matrix.

b: QoS

Quality assurance of network services is performed by measuring certain performance metrics such as:

Throughput: Instead of just collecting data, the authors in [104] developed a QoS monitoring module for POX controller. The main purpose of it is to offer QoS information or measurements for applications such as traffic engineering. In [104], the monitoring module polls statistics from ingress&egress switches to compute throughput by counting the number of bytes sent along the duration of that flow. Similarly, the authors in [106] use the path's first and last switches to measure per-flow throughput. Moreover, authors in [107] measure the flow rate by polling statistics from the first and the last switches. The difference between the current and the recent measurements over their time window is used to compute the flow rate. The authors in [185] use OpenFlow statistic collection messages to collect bytes and packets passed through switch ports. They are interested in computing the current transmission rate of each link by counting the bytes that pass the link over some time. Those computed values by the monitoring module are used by the forwarding module for load balancing when a new connection is received. In which, when a link utilization reached a certain threshold equals 80%, another path is chosen instead of the shortest path to avoid packet loss.

In [119] the flows utilization is computed as the count of bytes matched at the switch for each flow over its duration. If flows are overlapping (i.e., which is in real life), the utilization is accumulated at each checkpoint which means that utilization of the next expiring flow will be added to the checkpoint created by the last expired flow if those two flow overlaps. Even it creates no overhead, this approach cannot offer instant utilization of the network, and it depends on *PacketIn* messages, if no match is happening, which is a problem if the switch is instantiated with proactive flow rules. Also, the expired message may take more time for long-living flows to expire.

Delay or Latency: The authors in [186] propose an approach for link latency monitoring. It consists of two modules namely; *LLDP discovery* and *Echo monitoring*. The first module tests the link latency between two switches using Link Layer Discovery Protocol (LLDP) packets. For the directional test, the controller sends an LLDP packet to the first switch which already guides by the controller to forward that it the second switch as in Fig. 11. It has no rules to tell how to deal with this packet (i.e., or guided to forward such packets to the controller as in Fig. 11). Therefore, it send a *packetIn* message to the controller. By this process, the controller records the link latency between the two switches in one direction only, if the opposite direction is tested another process is performed. The second module is used to calculate the propagation delay from the controller toward a network switch by sending times-tamped Echo messages towards that switch. Upon reception of that packet, the switch returns it to the controller, see Fig. 11. By doing that the controller computes the link latency between any to switches. Similarly did the authors in [187] by exploiting LLDP packets for link delay estimation and using Echo messages to tell the propagation delay between the controller and switches.

In [86] implemented a link discovery approach similar to LLDP but with smaller packets. LLDP works as in Fig. 11 in its simple form. Authors used that discovery procedure to measure links delay and periodically repeat that task. In [174], the authors injected a packet into the network that loops along the test path and goes back to the controller to test the latency, and this is called *TTL* loop. Similar to all probe-based approaches, it needs to insert rules for matching those packet *TTL*, decrements its value and forwarding toward controller when $TTL = 0$. Similarly, the work done in [112], [113], [183], [195] use similar approach to compute the link delay. In [112], the authors use probes arrival delays and RTT (Round Trip Time) values to compute path delay. Using the same way, the authors in [113] measures the flow's end-to-end delay. Each probe packet travels the path from the first to the last switch and goes back to the controller. This process is repeated periodically for each monitored flow. Upon its reception, the flow delay is estimated after subtracting the RTT from the controller to both switches. In [106], however, the authors added a coefficient of proportionality varying from 0 to 1 and considered the Round Trip Time from the first and last switched toward the controller. Similarly, In [116] delay is measured using probe-based approach for sub-paths. Link delay is inferred from those sub-paths delays unless it is uncovered, it is tested similarly to previously discussed works.

The authors in [195] proposed a framework called *SLAM* to monitor path latency in SDN-based data center networks in order to detect high delay network segments. Authors use customized probe packet to trigger *PacktInt* notification packets toward the controller when a new unmatched packet is received in the network switch. Then, it measures the latency based on the arrival times of the *PacktInt* message to the controller

For path delay, approach used in *OpenNetMon* [104] injected packets that travel between edge switches along the same interested flow's path. The delay is computed as the difference between arrival (i.e., at egress switch) and departure (i.e., from ingress switch) of that injected packet. The difference of probe packets arrival and departure is also used in [114] to compute link latency. The authors in [115] utilize the same probe-based approach (i.e. similar to LLDP packets in Fig. 11) to obtain the link delay.

For testing the queue delay, the authors in [89] utilize the same probe-based approach. In which, a special packet is sent to the first switch which is configured to send it to the next one. The receiving switch sends the packet to the controller since it does not know what to do. The difference in this work is in the first switch, along with the output action, the tested queue is selected by using *enqueue* action (i.e., changed to *set_queue* in later OF versions). By that, queue delay could be tested according to that approach, as in Fig. 11 the queue's delay between *S1* and *S2*. Differently, the authors in [175], derive a queue delay model from network parameters such as queue buffer size, queue bandwidth, number of flows, link propagation delay. Then, an estimated average queue delay is obtained from that model and used to control the end-to-end flow's delay. To fulfill a specific delay requirement a flow(s) can be switched to a different queue when an upper delay bound is reached. The most interesting of this work is that most of the parameters used in that model is maintained by the controller or can be polled from switches. The propagation delay on a network link can be estimated by injecting probe packets at an earlier stage when no traffic exists. Authors assume that the queue delay is the dominant cause for network delay, the packet processing is neglectable and the propagation is constant.

Bandwidth: In [99], [100], the data rate is computed as the difference between sent bytes measurement in the monitoring interval. In [128], bandwidth is computed as a function of the transmitted bytes per time unit. The authors in [88] use that port statistics for link and path capacity estimation. A network topology that is maintained by the controller is used to query network switches to get transmitted bytes by each switch port. The difference between transmitted bytes between any two neighbor switches represents the bandwidth consumed by the link between them. Then, it is discounted from the maximum link capacity to compute the available link bandwidth or capacity. For a flow, the available path bandwidth is the minimum of all links estimated capacities along that path.

In [89], available queue bandwidth is monitored by polling queue statistic from network switches. Utilization of a queue bandwidth is computed from the difference between two consequence transmitted bytes readings over that time window. Available bandwidth is obtained by subtracting utilized bandwidth from what the queue is configured with. Similarly, the authors in [86] measure link utilization by considering transmitted bytes over the polling time interval (i.e., 5 seconds). This measurement is compared with the configured link bandwidth.

In [127] switches are used to compute link utilization in both direction (i.e. send/receive). The difference between the current and the last port statistics (sent/received bytes) is computed over the monitored interval.

In [90], [91] link utilization is measured by accumulating flows statistics that share it. Therefore, flows are grouped by that link and group table entries are polled to collect flows statistic that shares that specific link.

Loss: For packets loss measurement, the authors in [104], [106] compute it as the difference of flows packets statistics at both ingress and egress switches. The authors in [86] measure the link loss from port retrieved statistics. Over the conducted polling interval, transmit drop bytes are used to compute the loss rate of that link. In [99], the loss rate is computed as one value for the link in both directions by taking the difference of port sent and received bytes on the arc between the two adjacent switches over the monitoring interval.

In [181], the authors logically partition the network into different ring-based link-exclusive probe structure to test link loss rate and position the loss location. Rules are installed into switched to define such structure. Each probe structure is probed by a test machine that sends probe packets periodically based on instructions from the controller. The number, loss, and delay of received probes from those previously sent ones are reported to the controller by the test machines. Based on, and with statistic collection from switches, the controller can determine the link loss rate and position it within the network.

The authors in [96] aim to improve the accuracy of packet loss statistics by taking the effect of raw packets (i.e., non-data traffic) into account. It is applied only on link loss statistic since it does not count on flow traffics. Therefore, for any two communicating switches, traffic loss is computed as the difference of sent and received packets at both source and destination ports in the source and destination switches respectively. Similarly, the loss in non-data traffic is computed which is discounted from traffic loss to remove non-data noise in loss monitoring. For flow loss calculation, non-data traffic is not considered.

In [182], the authors used loss statistics from network switches and computed the MOS metric for VoIP QoE and see the effect of loss on that metric. So they periodically polled statistics from network switches, and then the link loss rate of each two adjacent switches is computed.

Table 5 shows those monitored QoS metrics against their tested entities. Flow monitoring over micro view of the network, which is more coarser in link based monitoring that gives a macro view of the network state.

c: FLOW SIZE

Large size flows consume network bandwidth and make it difficult for short living flows. In [94], the authors test their approach to tackle the problem of the hierarchical heavy hitter for identifying large traffics. Also, they assume that there is only monitoring rules and no actions that either drop or forward the packet. Authors of [129] use two random sampling, per packets and per bytes. Those sampled packets are sent to the controller for heavy flow detection modules that uses those sampled packets to detected suspicious heavy flows based on some threshold. Upon suspicious, count rules are installed within the switch to count the flow. In [130], the sampled packet is marked, so they not sampled by consequence switches. Also if suspicious flow's packets are traversing multiple paths, counter rules are installed in all those switches to aggregate them at the controllers as for one flow. These counters are polled periodically by the controller for the heavy flow detection module. *OpenMeasure* [92] use online learning to estimate per-flow size and identify expected large flows. By adapting rule installation, a list of flows and their sizes are monitored.

Table 5 gives a summary of QoS monitoring work in SDN that been reviewed in this survey.

In summary, we can make the following remarks:

- In measurements collection, there is always a trade-off between accuracy and collection costs (i.e., communication or TCAM memory). Therefore, the statistic collection should be adapted, and there is no place for static configuration.
- Another essential aspect that may affect measurements accuracy which the non-data traffic. Network management packets for link discovery, DHCP and many other that is produced or consumed by switches. For sure, these are counted in some statistics, especially link related ones. The authors in [96], [97] pointed out that problem and tried to estimate it and remove that noise from data traffic measurements. Also, control messages need be monitored besides the data traffic, either to obtain the volume of that traffic or visibility to network operators. For instance, the authors in [131] implemented a module for monitoring OpenFlow control messages In/Out of the ONOS [56] controller to allow the administrator monitors the control plane through web-based GUI and provides a history similar to logging in network systems. Monitoring control and non-data traffic is an open issue. Monitoring module should consider that issue.
- Switch selection for polling statistics need to be carefully optimized taking two things in consideration; flow-coverage and switch available resources. Polling all network element with fast frequency is not a good choice for monitoring even if in-band networks are used since more load are added to switches. Sharing the control with data channel will reduce the performance of the network if all switches are polled. Therefore, low rate measurements from all network location can be used in estimating the behavior of the network which could be used to adjust flows polling intervals.
- Accessibility is a significant task for autonomic QoS provisioning, since some functions such the analysis may not reside on the same controller or the station. To the benefit of already existing traffic analysis such as those used with NetFlow in legacy networks. In the SDN, it is a design principle to offer accessibility through northbound APIs. However, standardization of such interface needs more investigation.

B. ANALYSIS

The analysis function can obtain a series of time-stamped raw measurements (i.e., flow or port counters) or a preprocessed data such those in the form of QoS metrics or traffic matrices from the monitoring function. Further analysis of such data for the following:

1) DETECTION OF NETWORK QoS VIOLATION

Link congestion can lead to service degradation. The approach used in [127] monitors link utilization and produces a table of links utilization as an output of the monitoring process. That table is used as an input to the analysis phase to detects congested links. A simple threshold-based (above 70%) detection is used. Traffic engineering is performed to resolve that link contention by rerouting long-lived flows to other non-congested links. The reader can refer to [134] for a recent survey on congestion detection.

TABLE 5. QoS monitoring in SDN.

Ref	Approach	Objective	Timing	Polling interval	Controller	QoS metrics									
						thr		flow-size		bw		delay		loss	
						L	F	L	F	L	F	L	F	L	F
[92]	OF *StatReq*, adaptive rule installation	Reduce TCAM usage	periodic	–	POX				✓						
[88]	OF *StatReq*, special UDP packets	Bandwidth estimation	periodic	[0.5 ,100s], 0.5s most accurate	Open Daylight					✓					
[106]	OF *StatReq*	Balance overhead and accuracy	adaptive	[1s,30s]	POX		✓					✓			✓
[107]	OF *StatReq*	Reduce overhead and improve accuracy	adaptive	[1s,15s]	RYU		✓								
[114]	Random walk probabilistic probes	Congestion detection,Link delay	adaptive	Maximum probes/minute/flow	RYU							✓			
[89]	OF *StatReq*, Probe packets	Estimate queue delay and available bandwidth	periodic	every second	Floodlight							✓			
[113]	Probes	Path delay	periodic	–	–							✓			
[116]	Probes sub-paths	Measure link delay and reduce prob overhead	periodic	–	Pox							✓			
[183]	Probes	Measure link delay and reduce overhead	periodic	–	Floodlight							✓			
[115]	Probes with optimized interval	Measure link delay	adaptive	–	OpenDaylight							✓			
[86]	OF *StatReq*, Probe packets	Monitor QoS metrics	periodic	every 5 seconds	Ryu					✓		✓		✓	
[83]	OF Messages, switches with VoIP flows	VoIP flows monitoring	periodic	every second	OpenDaylight										✓
[176]	OF Messages.	Estimate TM, heavy hitter, Optimize TCAM usage	periodic	every τ seconds	–				✓						
[87]	OF Messages. Different switches querying algorithms	Estimate TM with Overhead	periodic	every 5 seconds	NOX				✓						
[181]	OF message, and probes packets	Test link loss rate and position the loss location.	periodic	–	–									✓	
[96]	OF *StatReq*	Improve accuracy by taking off non-data packets	periodic	every k seconds	Ryu									✓	✓
[90] [91]	OF message, aggregated group-based monitoring	Reduce communication Overhead	adaptive	default=1s, [0.5s,5s]	Floodlight					✓					
[98]	OF *StatReq*, optimized polling	Reduce overhead by optimizing polling cost taking controller into account	adaptive	[0.5s,3s]	POX					✓					
[110]	OF *StatReq*,sampling	Balance between accuracy and overhead,	adaptive	[0.5s,5s].	RYU					✓					
[103]	OF *StatReq*, adapt the polling interval using history of measurements	Reduce overhead	adaptive	[0.5s,5s]	Floodlight		✓								
[94]	OF *StatReq*, adapted monitoring rules	Reduce overhead by aggregating rules	periodic	M ∈ [1s,60s]	–				✓						
[99], [100]	OF port messages	cluster switches to reduce queries.	periodic	t_mon	Floodlight, NOS	✓				✓		✓		✓	
[102]	OF message	Reduce overhead, decision to use poll-single/all messages	periodic	–	–		✓								
[104]	OF *StatReq*, Probes	Reduce overhead	adaptive	–	POX							✓		✓	
[195]	Probe packets	Detect high delay in network segments	periodic	–	POX							✓			
[112]	Probe packets	Measure path delay	periodic	–	POX							✓			
[95]	OF message	Reduce communication cost	periodic	every second	Simulation									✓	
[109]	OF statistics and event messages	Reduce overhead	adaptive	[0.5s,5s]	Floodlight					✓					
[119]	OF *PacketIn*, *FlowRemoved*	Reduce overhead	–	–	others					✓					
[120]	Threshold-based push in OF 1.5	reduce *StatReq*	adaptive	every 1 second	Others		✓		✓						
[123]	sFlow based sampling	–	–	–	Floodlight					✓					
[127]	Link utilization is computed and pushed to controller	–	periodic	every 0.5s or longer	POX					✓					
[128]	OF *StatReq* and sFlow sampling.	Improve accuracy	periodic	every 0.1/1s, and N=10,N=256	Floodlight					✓			✓		
[129]	Random packet sampling, Then OF monitoring rules	Reduce overhead	periodic	Probability p	–				✓		✓				
[175]	Network model that can use statistics	Estimation of queue delay	–	–	Floodlight							✓			
[182]	OF *StatReq*	Monitor the effect of loss on VoIP quality	periodic	every P seconds	POX										✓

Moreover, massive bandwidth consumption needs to be detected due to the criticality of network resources to the QoS support. Elephant flows consume bandwidth and fill switches buffers which are a problem for a short-life flows (i.e., called mice flows) or delay sensitive ones as in VoIP flows. Therefore, detection and redistribution of them along

TABLE 6. Comparing features of the analysis functional component.

Ref	Input	Method	Output	Controller	Objective	Domain	Network	Control	SDN Plane
[127]	Link utilization table	Threshold-based detection (above 70% link utilization)	Traffic Re-engineering for long-lived flows	POX	Congestion detection	Single	Data Center	Single	Control
[150]	QoS real-time path information (i.e. bandwidth utilization, loss ratio, latency) and hop count	3-layer BPANN model	least loaded path, for load balancing	Floodlight	Load balancing	Single	Campus	Single	Application
[151] [152]	QoS network parameters (i.e. bandwidth, jitter, delay, etc.)	regression ML algorithm	predicted MOS value, adapted video parameters (i.e. resolution, frame rate)	Floodlight	QoE prediction	Single	Campus	Single	Application
[153]	Available network resources, video content, resolution	ML algorithm (random forest)	the best next level (i.e. in DASH video streaming)	Floodlight	QoE prediction	Single	Campus	Single	Control
[154]	VQM and SSIM	four ML algorithms (e.g.neural network, Decision Tree, k-NN and random forest)	MOS value for user perception prediction	Floodlight	QoE prediction	Single	Campus	Single	Application
[155] [158]	infrastructure (network and server cluster) statistics X	regression tree and random forest	Service metrics (i.e. response time and frame rate)	Floodlight	QoS prediction	Single	Enterprise	Single	Application
[159]	traffic matrix	Neural Networks	end-to-end delay estimation	NA	QoS prediction	Single	–	Single	Control
[162]	network QoS metrics and IT infrastructure KPI(e.g. CPU, RAM)	linear regression ML algorithm (i.e., M5Rules)	which KPI has the most impact on QoS	POX	QoS prediction	Single	Data Center	Single	Application
[163]	SSIM and VQM	Game theory model	MOS	RYU	QoE prediction	Single	Campus	Single	Application
[164]	network condition	ML algorithms	SSIM index	NA	QoE prediction	Single	Campus	Single	Application
[171]	application KPI and network QoS metrics	multiple linear regression correlation	application KPI metric.(i.e load time)	Ryu	KPI/QoE prediction	Single	Enterprise	Single	Application
[167] [166]	QoS values of video flows	Fuzzy Logic-based quality prediction	video PSNR, then adaption by the service provide or the controller by rerouting	NA	QoS Prediction	Single	Campus	–	–
[168]	network traffic measurement	K-NN algorithm	future network measurement (e.g., # of packets, flows)	NA	Traffic Prediction	Single/ Multi	–	–	Control
[169]	link matrix (i.e. composite of links latency, rate, and bandwidth)	Markov model	link popularity matrix, Congestion avoidance	POX	Congestion Prediction	Single	Data Center	Single/ Multiple	Control
[170]	flow monitoring measurements	Poisson Shot Noise process	Traffic Matrix	NA	Traffic Prediction	Single	Data Center	Single	Control
[160] [161]	fixed size or a sliding window of last traffic matrices at t	LSTM-RNN	a traffic matrix at $t+1$	POX	Traffic prediction	Single/ Multi	Enterprise	Single/ Multiple	Application
[156] [157]	Collected data and Preprocessed data (i.e. feature extracted)	Forecasting at first, then DT algorithm is used to do SLO violation prediction	Forecast to predict Y matrix of selected features at t+1, that fed to DT, SLO violation predicted and SLA enforcement issued	OpenDayLight	SLA violation prediction	Single/ Multi	Cloud	Single/ Multiple	Application

the network or taking the appropriate action defined by the QoS policy. The authors in [135] focus on visualizing such flows for the network operator. In [144], the authors put more attention on the adaption of the configuration of detection or decision thresholds. The authors in [145] used a learning model to detect heavy hitters. Heavy hitter consumes a large portion of network capacity that could be to service nature or suspicious users. A predefined mark list by the network administrator of known heavy hitter is provided to the model in conjunction with flows statistics. Two threshold values,

$\triangle_1 < \triangle_2$, is used to compare flows rate with history values. If flows rate is above $\triangle_2$, it is considered as a heavy hitter and added to the predefined list. If the rate between the threshold values, it is considered suspicious and more focus put on that flow. Feedback is reported to the monitoring unit to increase the sampling interval or install more fine-granular monitoring rules for that flow. Flows can be removed from the list if their behavior becomes normal, in which negative feedback is observed, and the flow is removed. Similarly authors in [136]–[143] proposed schemes to detect and mitigate elephant flows. The reader can refer to [146] for more information about elephant flow detection methods.

The authors in [132] proposed a predictive model for monitoring SLA maintenance for cloud tenant applications. They used historical data collected from the flows of the application and use it as an input to a model that compute two values: the first one is the mean throughput and the second is a scaled score for application behavior. A forecast engine receives these values as input. It periodically monitors the flows of the application and uses the inputs from the model to compute a metric value for volatility detection. If this value above a certain threshold the rate of the application is limited.

Author in [156], [157] utilize ML in SLA management in NFV and SDN. They proposed an SLA enforcement architecture. It consists of three parts, the data collector (from virtual machines, functions, and switches), data processing (i.e., preparation and feature extraction) and the smart engine which, based on the collected and pre-processed data, can predict violation in SLA. Prediction depends on a forecasting module that takes the processed data and tries to predict or expect future values. Then, the prediction module can identify which SLO expected to be violated. Then, it issues a warning message to the administration layer, and the SLA enforcement module is notified to do proper management action such as allocating more resources.

2) TRAFFIC PREDICTION

The authors in [160], [161] use LSTM-RNN (Long Short-Term Memory Recurrent Neural Network) to predict future traffic. At each point of time t, the predictor takes as an input a fixed size sliding window of the last traffic matrices and uses it to produce the traffic matrix at $t + 1$.

Similarly, the authors in [165] used Bayesian-based classifier for traffic prediction. Moreover, the authors in [168] use a trace of network traffic for a long time interval (i.g., for one day), and divide into smaller lags or periods to form the prediction data set. During each lag, at least one network measurement is obtained. The measurement contains the number of packets, bytes, and flows. This measurement value(s) is matched with those values in the generated dataset. A K-NN algorithm is applied to find the closest match values (i.e., from the history data set, $k = 1$). Then, the value of the next lag time is considered as the predicted value. Their approach is similar to what been used in road traffic prediction, in which the number of cars at next time interval can be predicted from daily historical data of the traffic at that point of time within the week. The authors in [170] propose flow based Traffic Matrix predictor called *FLAME*. They made two assumptions: flow arrivals rate is a Poisson process, and flows rate (i.e., shot) is independent and identically distributed. Based on Poisson Shot Noise process, and the monitoring measurement, it estimates the traffic volume for the next 10 seconds in future.

The authors in [169] propose a QoS management framework in Data Center Networks, called *Horizon*. In their work, they used different link matrices. A link performance assessment matrix is computed and updated periodically for all network links. For each link, the assessment value computed as the weighted sum of its latency, rate, and bandwidth. Based on that matrix, they compute links transition probability matrix which is used to build a multi-step Markov process transition matrix. The Markov transition matrix is used to compute another link popularity matrix. Link selection and popularity matrices are used for path selection to avoid congested links. Based on that, It considers the hotlink state when the path is selected. Flows can span different paths to balance the network load. Their proposed work is evaluated on a flat tree topology within the POX controller and compared with the ECMP protocol.

Moreover, authors in [150] proposed a network load balancing based on SDN and ANN. Based on collected QoS information (i.e., bandwidth, utilization, loss ratio, latency) about the network state. The controller sends that path(s) load information to a load balancer. It uses the QoS data and path hop count as features of each path and as an input to a BPANN (Back Propagation Artificial Neural Network) model. The balancer predicts the next least loaded path, which used by the controller to transmit newly arriving data flows.

3) PREDICTION OF QoS

A reactive action upon the detection of service quality degradation is essential. However, for autonomic QoS support, the network system needs to go beyond that. Prediction is a proactive procedure the system follow before any problem occurs [148]. For instance, the authors in [149] uses prediction of network utilization for future reserve resources to satisfy users or cope with high priority requests. The authors in [159] investigated the usage of Neural Networks techniques for building models that estimate the network performance. The network is simplified as a black-box which takes traffic matrix as an input and produces the average end-to-end delay estimation as an output of that model. The accuracy of different neural network models is tested against various network characteristics (i.e., size, topology, routing, traffic intensity).

In [162], performance indication data is collected from the virtual network (statistics of port and queues), traffic (e.g., delay, loss) and IT infrastructure (e.g., CPU, RAM) to discover the correlation between KPI and QoS metrics. Then, a linear regression ML algorithm (i.e., M5Rules) to quantify which KPI has the most impact on QoS.

The authors in [155], [158] estimated end-to-end service metrics based on statistics collected from the network (i.e., port-based) and back-end clusters (i.e., the infrastructure of video streaming service). It is expected that degradation on the infrastructure will affect the end user. Based on that, and by using input statistics from the infrastructure, the service level metric (i.e., frame rate, response time) at the end user could be estimated. This regression model for approximating service level metric is solved using supervised learning by the authors. From the server side, they collect statistic about CPU utilization, RAM utilization, disk I/O and so on. Moreover, from the network side, port statistics are collected such as total sent/received bytes/packets per ports along the whole path. These statistics are treated as a bag of features used with two learning methods: regression tree and random forest for the estimation process. Later, used features are reduced to use only the network statistics for in the estimation process. The accuracy of the achieved results was close to when full features are used. Authors mention that service-level properties are encoded in the network-level statistics in which end-to-end service metrics can be estimated from network-level statistics.

4) PREDICTION OF QoE

The authors in [151], [152] proposed schemes to predict MOS (i.e. Mean Opinion Square) value for a video streaming service. They used Machine learning regression algorithm to do that. Based on monitored parameters such as delay or bandwidth and the estimated MOS, the video parameters such the frame rate can be dynamically changed by the controller without interrupting the user. In [153], the authors optimize the QoE based on five measured video quality that may affect the user satisfaction such as bit-rate, stall number, switches between levels (i.e., degradation in video quality). First, they utilize machine learning to estimate the best next level (i.e., in DASH video streaming) based on the available network resources, video content, and device resolution. Random forest is used as the estimation algorithm. The prediction phase is performed when the controller detects a change in the network that can affect the current or the selected level. Then, during the second phase, a decision is made after comparing the estimated level by the machine learning and the currently used level. The decision algorithm tries to maintain QoE by smoothing the degradation of video quality when the estimated is lower than the current level. The authors in [154] utilize four ML algorithm to predict MOS (Mean Opinion Score) value namely; DT (Decision Tree), neural network, k-NN, and random forest. This value usually used for estimating QoE for video streaming services and can indicate user satisfaction level. The prediction is built based on values of two parameters: VQM (i.e., measure the effect of video distortion) and SSIM (i.e., measure structural similarity). Based on Pearson correlation coefficient (r) and Root Mean Square Error (RMSE) as an evaluation metrics between the predicted MOS values (i.e., using ML algorithms) and subjective MOS value (i.e., collected from participant-based video quality evaluation). Authors show that the Random Forest algorithm was the best in predicting user perception. In [163] authors, modeled the relation of SDN network, DASH streaming and user perception or the MOS value as a game theory in which they proposed a solution to minimize a cost function by minimizing the VQM value and decrease levels variation.

The authors in [164] used ML to predict SSIM-based video quality considering network condition into account. Based on, the predicted SSIM index, the required video quality level can be adapted. Correlation between network condition and the SSIM index is performed. The appropriate video codec is chosen to offer the minimum QoE users need [164], [167], [171].

The authors in [167] propose a QoE agent that uses QoS values of video flows to assess their impact on user QoE. It uses a Fuzzy Logic-based quality prediction model that maps QoS parameters (i.e., bandwidth, packet loss) into QoE (i.e., PSNR) [166]. Then, the predicted QoE information is sent as a feedback to the service provided using SIP protocol. It enables the provider to improve the service level in which an adaption process can be performed.

Application KPI/QoE metric prediction using network data in SDN is investigated in [171]. Load time KPI metric in web-based mapping (e.g., similar to Google maps) application is considered. The SDN controller receives history time-stamped KPI metric values through JSON file, and the network metrics (i.e., Delay, Bandwidth Capacity, Rx Load, Tx Load and Packet Loss) is obtained from network statistics. From those metrics (i.e., KPI and network metrics), a dataset is constructed. Then, they developed a model that uses the network metrics as an input feature while the KPI metric is an output that is predicted by the machine learning algorithm. This model can be used in future flow installation decision in a way that improves the application performance. Different machine learning algorithms can be used such as RNN, SVM, or DT (Decision Tree). However, the experimental work shows only a multiple linear regression is used to characterize and correlate the relationship between the network QoS metrics and the application KPI metric (i.e., load time in this case). They suggest using those sophisticated ML algorithms in the future.

The authors in [133] proposed a video surveillance system in which two modules are used. A classification module to detect traffic type that traverses the SDN network and report the criticality of such flows (i.e. multimedia or video traffic). And an AI-based module that tells the controller what to do to guarantee QoS for video transmitted flows. Such as the required resources (i.e. bandwidth, prioritization, or routing) to offer a satisfying service level for the users. Based on the suggested action by that module, the controller can do the required configuration. Similarly, the authors in [147] did.

In summary, we can make the following remarks:

- Many of the reviewed research approaches use simple threshold-based analysis of the QoS performance. However, this is not helpful for self-adaption to network changes and autonomic support. This function should take more space. Learning aspects should be applied such as the use of machine learning techniques. Some research effort exploits the ML in detection and most of them in prediction for future trend in traffic or user satisfaction. More attention should be applied to the usage of online analysis which adds more challenge to reduce the time of training when these techniques are used.

- Moreover, we notice that most of the research uses static policies or configuration. Modification only occurs once, either due to the specific scenarios applied in such research or it a design choice to avoid that. The autonomic QoS provision should get benefits from experience in either human format converted to polices without ignoring the time dimension from its application. Alternatively, machine defines policies that are inferred from analysis of the history information collected from the network.

C. PLAN

The adaption of network behavior by the QoS management is essential to cope with new QoS demands or services quality enhancement requests (e.g., from the analysis unit/function/module). The management may search for other QoS-compliant routes or allocate more resources. In this section, we discuss the two procedures used to serve QoS demands in SDN.

1) QoS-AWARE ROUTING

In QoS provisioning, routing plays a significant role to guarantee the required QoS demand(s). It tries to find an alternative path(s) that satisfies QoS requirement for flows.

In the SDN architecture, the controller maintains an abstract view of the network. Most of the SDN controllers contain a module that provides forwarding rules installation service. For instance, the Floodlight [52] controller implements a forwarding module, but it does not provide routing functionality. However, in the Beacon controller [50], a routing module implements the Shortest Path(SP) routing or what is called the Dijkstra's algorithm [188]. That module performs the computation of paths for all pairs. Due to the central view and up to date state of the network, it is easier to compute the shortest path within the SDN. The dynamic changes of network state and flows arrival rate affects the routes computation frequency. For instance, routes may be computed once in a proactive mode (i.e., off-line routing or planning) with the advance knowledge of all flows, and their demands are fixed or merely changing. However, in online routing, routes are computed whenever no path is found to forward a flow's packets. We consider this routing in this survey.

In QoS management, the problem is more complicated since the routing algorithm should satisfy the flows requirements. Otherwise, the service quality may start degrading if the used routes are congested. Based on the QoS requirements, different routing problems can be defined:

a: SHORTEST PATH (SP)

In shortest path variant algorithms, a single QoS metric is considered while optimizing path computation. In [190], the proposed approach implements a modified version of the shortest path algorithm in the ONOS controller. It selects the least load shortest path to route QoS traffic. While in [189], the authors use the shortest path for non-QoS traffic with the hop count as the path cost. For the QoS traffic, the algorithm computes the shortest path from a residual resources graph. Links that do not satisfy the new bandwidth requirement are removed from the original network graph. The outcome is the shortest path that satisfies the bandwidth requirement.

In [191], the authors used an SDN-based approach that intercepts RTSP/RTP streaming setup packets and exploits them to setup QoS routes for their video flows. They used the shortest path with link weights. Links that have higher weight have a priority over others to be chosen in the QoS route setup. The link weight depends on the utilization of that link. They used statistics collected from OpenFlow enabled switches to estimate the utilization of links. The network state information helps in adapting the QoS routing. This work depends mainly on the accuracy of statistics collected from the forwarding nodes, which is a trade-off between accuracy and extra overhead over the network that should be maintained carefully by the monitoring module. Similarly, in [205], the authors utilize a flow classifier to get 5-tuple information about each incoming flow. The classifier gives the source/destinations IPs, ports and the used protocol (i.e., RTSP). The routing module produces a set of available paths and their costs. The cost of a link is computed as the sum of received and sent bytes by the switch's port. Based on that cost, the shortest path is used. In [206], the link cost metrics are computed based on received network state information. Specifically, the port sent and received bytes at both arc nodes. Then, data and loss rates of the link are used with its configured capacity to compute the link cost value dynamically. After that, the routing module computes the shortest path that has the minimum cost. Authors assume that users augment the routing framework with the highly prioritized traffic (i.e., such as video streaming) information (e.g., protocol type, port numbers) which helps to classify flows.

The routing in [207] is a combination of the shortest path and the shortest-widest path. It finds the widest path and compares its bandwidth required by the flow. If it is less than what is feasible for the flow, then it is chosen as the bandwidth the flow will have. Otherwise, it uses the flow's feasible or required bandwidth. Based on the chosen bandwidth's value, a run of the shortest path is performed to find the path which can offer that bandwidth demand. By doing that, they attempt to save more rich paths for the future if the required bandwidth is small and another path can serve it. In general, the approach finds the shortest bandwidth feasible path. The approached proposed in [208] uses bandwidth information from the monitoring unit for each port. For a QoS flow, it checks if there is an alternative path with better bandwidth than the shortest path. If such a path is found, use it for that flow.

The work in [209] applied off-line and online routing. Routes are computed (off-line) when the network is initialized and recomputed when there is a change in the topology. When a new QoS request arrives, the online routing takes place. It first checks if an existing path (i.e., from off-line computed paths) can satisfy the bandwidth demand. Then, the feasible shortest path is chosen. If no off-line bandwidth sufficient path exists, the shortest path is used to get a path from the residual graph where links below the bandwidth demand are filtered out. If such a path could not found, the request is rejected.

In [211], the proposed approach monitors the link bandwidth utilization. Upon detection of link congestion (i.e., utilization over a certain threshold), the routing module is invoked to find alternative paths that satisfy flows bandwidth requirements. The shortest path with available bandwidth is chosen as the new route to resolve such a situation.

In [210], the authors proposed an algorithm to route layered SVC-based video streaming in SDN. They classify the traffic into three classes; the base layer of the video as lossless QoS, the enhancement layer as a lossy QoS and the other traffic as a best effort class. The routing algorithm may choose different paths for the video layers. They used a modified version of Dijkstra algorithm with a weighted metric composed of the mean link loss and delay. Upon reception of a client request for a video service, the controller finds a path toward the video server and pick a ToS (Type of Service) value for each video layer. Then, the server uses these values when sending the video layers packets (i.e., tagged with agreed ToS value). By doing that, the controller knows which layer these packets belong to by just inspecting the one-byte ToS value. This method helps in avoiding the usage of deep packet inspection. However, ToS value may be changed by other parties the packets traverse their networks.

The shortest path algorithm is used extensively in the literature. A well-designed QoS-aware cost function may add to the fast and simplicity of path computation provided by the shortest path algorithm. Especially, with the up-to-date network information received from the monitoring unit as discussed in Section IV-A.

b: (MULTI-)CONSTRAINED SHORTEST PATH (CSP AND MCSP)

The shortest path gives an optimal path according to a specified cost metric (i.e., in QoS optimal path, the cost is a QoS metric). However, and in many cases, other QoS metrics may need to be considered and maintained within acceptable bounds as it is specified in the SLA contract. Therefore, another form of QoS routing problem is reported to find an optimal path according to one QoS metric while controlling another within required bounds.

In [212], the authors optimized the path selection for videoconferencing flows based on its delay as the QoS perused metric. They computed the cost metric as the weighted sum of lost packets and delay. Their objective is to adapt between the delay and the cost while computing paths for those flows. In [213], the authors propose an architecture to support QoS video flows in SDN. The authors differentiated between two layers in the SVC encoded video. A base layer that should receive a particular QoS in which packet loss is forbidden and an enhancement layer that is classified as the best effort traffic. The approach directs the base layer traffic to non-congested routes due to its primary effect on the video quality. The process is as follows: the streaming server contacts the SDN controller with a QoS request message, the SDN controller configure the path of the QoS traffic according to the QoS agreed contract and return a QoS id for the streaming server which starts streaming. Upon reception of queues statistics or network error, the SDN detect that the path is congested so it cannot handle QoS traffic. Therefore, the rerouting algorithm works to find another path with the appropriate capacity even if it is longer than the shortest path (i.e., the best effort always uses the shortest path) but with a delay or length that is tolerable by the streaming application or less max value. After that, flow modification is sent to nodes to configure the new path. The result in their work shows that when a new UDP traffic is entered the network, and by rerouting, the QoS traffic does not suffer much. The model formulated by the authors consider packet loss as the primary constraint for path selection in which a zero loss should be achieved. In this approach may not be applicable for another type of traffic such as VoIP which is tolerant toward packets lost and sensitive for delay or jitter. Similarly, in [214] authors divided the video stream into two layers: a base layer served as QoS-traffic that should have higher priority and an enhancement layer as a second-level QoS traffic. The rest of the traffic type is served as the best effort which always takes the shortest path (i.e., least hop-count). However, they propose to choose alternative paths for QoS traffic. They performed optimization on route selection and formulated the QoS-aware routing as a constraint shortest path problem (CSP). They showed that using that dynamic routing for QoS traffic improved the overall video streaming quality. The packet loss measurement is defined similarly to [213] but with separation of the QoS traffic into two levels. This work depends on the precise values returned from the switches (i.e., the accuracy of information collected by the monitoring function or module discussed in Section IV-A) to estimate the packet loss and delay variation of each link which is used to compute the cost metric that based on route calculation is performed. Also, they solve the CSP problem two times to find routes for 1-level and 2-level QoS traffic respectively which may make it run slower.

The authors in [215] divide the SVC encoded streaming video into two layers, loss-less QoS flow for the base layer, and lossy-QoS for the enhancement layer. The rest of network traffic is considered as the best effort which follows the shortest path. They used the loss as the quality of service metric due to the sensitivity of SVC base layer to it. Therefore, loss-less traffic is routed first compared with enhancement

```
procedure LARAC($s, t, c, d, \Delta_{delay}$)
  $p_c$ := Dijkstra($s, t, c$)
  if $d(p_c) \le \Delta_{delay}$ then return $p_c$
  $p_d$ := Dijkstra($s, t, d$)
  if $d(p_d) > \Delta_{delay}$
    then return “There is no solution”
  repeat
    $\lambda := \frac{c(p_c)-c(p_d)}{d(p_d)-d(p_c)}$
    $r$ := Dijkstra($s, t, c_\lambda$)
    if $c_\lambda(r) = c_\lambda(p_c)$ then return $p_d$
      else if $d(r) \le \Delta_{delay}$ then $p_d := r$
                                   else $p_c := r$
  end repeat
end procedure,

where Dijkstra($s, t, c$) returns a $c$-minimal
path between the nodes $s$ and $t$.
```

FIGURE 12. LARAC Algorithm; s, t are the source and destination, c is the optimized cost function, d is the delay metric, Δ_{delay} is the delay constraint value, p_c is the shortest path with c as cost, p_d is the path with d as the cost, λ is the Lagrangian multiplier that yield the new cost c_λ [217].

and best effort. They attempted to minimize the weighted sum of the route length for base layer traffic and its packet loss rate. In this work, the authors evaluated an approach like the work in [213] where the enhancement layer is considered the best effort traffic. Then, they modified the problem to consider enhancement as QoS traffic with a possibility of packet drop. They showed that the second modification of the problem improved the video quality which is expected since extra frames can be received at the receiver side better than when they considered them as best of effort traffic.

In [216] authors define two types of traffic, a QoS traffic, and the best effort. They primarily used the shortest path for both traffic. However, when congestion is detected, and the requirement for QoS flow cannot be satisfied, a routing optimization algorithm starts working to find a path that satisfies the QoS metric requirement. They use the delay variation as the metric. They modeled the problem as Constrained Shortest Path and used LARAC (Lagrange Relaxation based Aggregated Cost) [217] algorithm to solve it similar to what is in Fig. 12. The algorithm tries to minimize the cost function while keeping path's delay below a maximum constraint value. The cost function calculated as a weighted sum of link delay variation and packet loss. Their work uses QoS routing only when congestion has already happened which may affect flows that have a short lifetime.

In [218], the authors classify the traffic as multimedia and data traffic. Multimedia traffic always routed toward QoS guaranteed routes. The other data traffic follows the default, shortest path computed by the SDN controller. They used the LARAC algorithm for computing the routes for multimedia flows dynamically, in which congestion and delay are used as the routing decision metric. They constrained the routing problem with predefined delays. The problem with the shortest path is it does not consider the state of the network and always uses the least-hops path to the destination even if the network nodes are congested. It will eventually lead packets drops and many retransmitted by end entities in reliable communications (e.g., TCP). Like previous works, their work depends heavily on statistics collected from forwarders to calculate links cost. Also, they consider a predefined 70% utilization threshold of the link capacity. They consider the link is congested if its utilization is above that value. Links with less than 70% utilization their congestion measurement is zero, which link delay as the main factor for the routing decision. By doing that, almost congested links can be chosen if the statistics packets got dropped or update interval is long. Another concern is that they put the values of the delay in the cost calculation equals to 1 (i.e., they said it is because the current OpenFlow switch implementations do not have any support for collecting delay related statistics) which make the algorithm works based on hop count when the links are not congested as its used by the shortest path algorithm.

In [219] authors propose an SDN architecture support QoS applications requirements. The architecture contains four modules: Resource monitoring, Route calculation, resource reservation, and call admission control. They define two traffic classes, a priority traffic which require bandwidth guarantee (e.g., video traffic) and best effort traffic that represent the majority of Internet traffic. They used a modified Dijkstra routing algorithm that finds the shortest path that has sufficient bandwidth for the QoS flow. By using bandwidth as the constraint, their goal was to minimize the delay QoS metric as it is the most critical for multimedia applications. To avoid degradation of best effort traffic, they used weight metric for links that compute how much-utilized link is. They attempted to avoid those highly utilized link by routing best effort traffic on links less utilized or have more bandwidth. By this, they avoid congestion before it happens (the threshold used was 80% link utilization). The reservation module used Of-Config protocol to configure interface queues. The call admission control units are supposed to reject QoS request that cannot be satisfied and feedback users with that, however, the authors admitted that they did not implement this module yet and northbound API are not used and delay it for future work. Also authors in [225] model the routing as delay constrained least cost problem. They solved it by using kLAM approximation.

In [195] authors targeted video steaming over SDN. They divide the video stream into two layers, 1-level QoS is the base layer which has high priority and intolerant to packet loss or delay variation. Moreover, another 2-level QoS which is the enhancement layer. The third type of traffic is the best effort which is the majority. The authors propose a reroute approach for the base layer traffic (1-level QoS). The routing module computes a feasible path if this path satisfies the delay variation constraint and has a bandwidth that is enough to add the 1-level traffic to it, then the 1-level (base layer) is flow is rerouted using this selected path. Otherwise, if the bandwidth is not enough, the enhancement layer is rerouted, and the base layer stays using the shortest path. However, if the delay variation constraint is not satisfied, none of the base or the enhancement layer is rerouted.

Previously, only one metric besides the cost is considered. A flow may require more than one QoS metrics to be within specific values or constraints which makes the path computation process more complicated and time-consuming. In [192] authors built their work based on the work in *VSDN* [193]. They added reliability as an extra constraint besides those previously used (i.e., bandwidth, delay, and jitter) in *VSDN* for video traffic path computation. Therefore, links are labeled with these four metrics and reliability has high priority in path selection. They argue that they modeled the problem as Multiple Constrained Shortest Path (MCSP) and used A*Prune [226] algorithm for solving it. They showed that *VSDN* could support a low number of satisfied requests when a high-reliability value is requested compared to *RVSDN*. They did not show in their work mathematical computation of the cost metrics, or the optimization process. However, this work is interesting in which it adds reliability as a constraint for path selection.

In [227], the authors define a deterministic network model to estimate the maximum delay of the network and the delays of link's queues are estimated. For admitting any new flow on a specific path, they compare with the network parameters. For each flow, a service class reveals its QoS requirements (i.e. max delay, rate). For instance, to admit the addition of a new flow to an existing path, the maximum delay required by its class is compared with the current path delay. Thus, for other demands such the rate parameter or burst value for instance. The problem is modeled as Multi-Constrained Shortest Path and solved as a mixed integer program.

The authors in [228] proposed a way to maximize the aggregated QoE for all uses and services flows. They used session configuration information received from a SIP server to inform the controller of those sessions will be established. The most interesting, they estimate the MOS value of each of the traffic types (i.e., audio, video, data) using the loss QoS. However, they used end-to-end loss and the delay QoS to compute the MOS value for audio traffic. They used queuing model (M/M/1/K) to compute the average delay and loss probability for each node. Their work shows significant performance (i.e., cumulated MOS) over the shortest path when the number of flow requests is increased. They modeled the routing as Multi-Constrained Shortest Path problem, and they used Integer linear programming implemented by IBM Optimization Language to solve it.

c: MULTI-CONSTRAINED PATH (MCP)

In many cases, it is not necessary to find the shortest path, but the chosen path should cope the certain constraints to fulfill QoS requirements.

In [229] a flow scheduling module is responsible for satisfying flows QoS demand. They assume that each flow has QoS requirements represented as a vector of delay, loss rate, and bandwidth values. That module uses a QoS-ware routing to find the best fit path that satisfies those metrics. The problem routing is a modeled a multi-constrained path problem. They proposed an algorithm to find the best QoS path by using a simulated annealing technique. The algorithm starts by finding the shortest path. A cost value is computed as the sum of weighted satisfaction ratios of each required demands. It adjusts the weight value by the miss rate of that QoS demand. It means higher weight for demands that less satisfied in the past. It tries to balance the satisfaction of those demands when choosing future routes. The algorithm continues to find a neighbor path(s) for $T = 16$ iterations (i.e., the stop threshold). The candidate neighbor path replaces the shortest path if its cost is smaller or the simulated annealing probability of the costs difference is high. It makes the chosen path does not always stick with the least cost local path (i.e., they use probability to avoid local optima). Simulated annealing depends on the design of the probability function and the number of iteration used to abort the search process.

In [230], the authors consider critical flows' routing in real-time systems. Such flows have tight delay requirements in which packets received after a deadline is useless. Critical flows are prioritized based on their sensitivity to the delay requirements. For such path assignments, each flow demands two QoS metrics, delay, and bandwidth. They formulate the routing problem as Multi-Constrain Path (MCP) in which the path of a critical flow should cope with delay and bandwidth demands or constraints. To find a solution, they used relaxation in which one of the two constraints is relaxed at a time to find a heuristic solution. For instance, the proposed algorithm relaxes the bandwidth constraint and to find a path that complies with the delay constraint and vice versa. If both attempts could not find a solution, the MCP has no solution and the algorithm return no solution is found.

d: MULTI-PATH ROUTING

Previously, flows follow a single path. The routing module cannot benefit from existing paths to spread the flow's packets. In [196], non-video flows are routed using conventional method (i.e. shortest path). They used multiple description video coding (i.e., two descriptions) and split the video stream into multiple sub-streams, then apply multiple path routing on them. The routing algorithm finds two paths for the two video descriptions that belong to the same video flow. The used metric is the video distortion in which the routing algorithm tries to minimize it. To solve the routing problem after formulating it, they used a sub-gradient-based algorithm to solve the multi-path problem.

The authors in [238] proposed schemes to offer guaranteed network resources for cloud users. They mainly focus on the implementation of their approach to be in the OpenVSwitch by exploiting multi-path support. They define a QoS metric consist of the sum of the ratios, the minimum values required by the user over the real measured ones. They belong to three metrics that are defined in the SLA which are the bandwidth, the delay, and the number of hops (i.e., path length).

Therefore, whenever any of these values do not meet what defined in the SLA, an alternative path is chosen for the QoS-flow. They define two types of flow, QoS-flow, and best effort. Even if the authors used OpenFlow compliant switches, their approach was not clear about the role of the SDN controller. Also, this highly dynamic change of the route due to nonacceptance of one of the three metrics they consider will cause high fluctuation of the QoS traffic between different flows. Moreover, they do not consider queue occupation in their metric which may lead to loss of packets.

In [197], the authors implemented two techniques to provide QoS in SDN: differentiated service (*DiffServ*) and multi-path routing. For the *DiffServ*, they classify the application's traffic based on source IP address into three traffic classes; high bandwidth-demanding video traffic, low-delay interactive audio/video traffic, and the best effort traffic. The routing model chooses an optimal path for new flows from a database of up to date optimal computed QoS paths. Then to achieve bandwidth and QoS of parameters, *DiffServ* is programmed on the OF switched. Each output interface has multiple queues that packets are enqueue based on the traffic class it belongs. The QoS of these queues (such as Max or min rate) is used to guarantee the QoS requirement such as bandwidth by using these queues. Authors compared their work by another two approaches that implement only single path routing and with/out *DiffServ*. The proposed approach showed better Response time and throughput compared to single path routing.

An application-aware multi-path flow routing for SDN (AMPS) is proposed in [198]. Authors utilize ML-based flow characterization with application prioritization to do the path-flow assignment. An ML-based classifier (i.e., C4.5 Decision Tree) is used to classify flows instead of using DPI. The controller uses the first 50 packets to extract flow features. Then, based on the produced flow class, the controller uses Yen-K-Shortest Path [199] algorithm to find K feasible paths. Based on the path cost, and the application requirements (i.e., bandwidth and delay), a path is chosen. High priority flows statistic is collected to assess if the QoS requirements are met. Otherwise, the path computation process is fired up. In AMPS, a flow with the same source and destination could be routed using different paths, since the application priority with classification plays a significant role in path computation.

e: AI-BASED ROUTING OPTIMIZATION

In [200], a QoS-aware adaptive routing (QAR) is proposed based on Reinforcement Learning (RL). RL is an ML-based technique that uses information from the system to re-adapt its behavior. An RL-based agent receives the state of the system, and a reward/punishment based on its actions. It tries to increase the long-term reward/revenue by exploiting its previous actions while exploring new ones. Authors designed a QoS-aware reward function as a tunable composite of QoS-based metric (i.e., link and queue delays, link loss, bandwidth) functions. According to, a value close to one is preferable while the harmful effect of an action (i.e., on other switches operation) gives negative values decreases the reward value. A quality function keeps assessing the expected rewards received from actions within the current state. For a decision to be taken (i.e., chose next hop), the most quality returning action is chosen. The algorithm keeps updating its quality knowledge-base (i.e., state-action $\rightarrow$ reward table-like) using past chosen actions. This approach tries to find a feasible path according to the flow QoS requirements. Deep Reinforcement Learning (DRL) is investigated for routing optimization in the SDN-based network by the authors in [201]. A trained DRL-based agent is used to provide a one-step routing configuration. Moreover, DRL is used in [202] to regenerate link weights based on network state (i.e., Traffic Matrices or load). Thus, based on the trained DRL-based agent, the computed paths optimize the network delay.

In [160] a neural network-based routing is proposed, called *NeuRoute*. *NeuRoute* uses Deep Feed Forward Neural Network (DFFNN) to learn and match demands with routes. It collects outputs or path decisions history from an already running heuristic-based routing algorithm (i.e., for learning), combines them with the network state and the predicted traffic matrix (i.e., as in [161]) as an input to train its neural routing network. The time it takes to collect samples or to train the neural network is a performance decision. The designer should make that decision. The network state consists of all links costs and their available capacities at time t. Once the trained model is ready, it can be used to route new arriving flows. Therefore, based on the traffic matrix and the network state as an input to the model, it can produce an exact route as an output.

A genetic-based routing algorithm for enhancing video streaming in SDN networks (i.e., called GA-SDN) is proposed in [203]. Video flows and the level of network links utilization (i.e., uses ports stat) are inspected periodically. Whenever link congestion (i.e., threshold-based) is likely to happen, *GA-SDN* tries to find a better path for video flows. *GA-SDN* reassembles the network graph of the path as a chromosome. The path cost equals to the fitness function of the chromosome. The fitness function takes one of two values: zero if not feasible, or $1/path(cost)$, which means increasing the path cost will decrease the fitness value. The proposed approach shows better PSNR and throughput performance values compared with the Bellman-Ford [204] routing algorithm.

The authors in [231] introduced a QoE-centric routing. They assess the effect of network condition (i.e., delay and loss rate) in the QoE values. QoE requirements are passed to the controller, i.e., northbound API. According to those demands, they formulate the routing as a multi-constrained problem. They used an Ant-Colony meta-heuristic algorithm to find a solution. The algorithm tries to maximize the flow(s) QoE value (e.g., MOS) and accommodating flows constraints.

The authors in [232] surveyed QoS-aware routing algorithms could be used in SDN. Specifically, the delay constrained protocols. They tested their performance using different criteria such as the topology size, the delay constraint. They found that LARAC had the best performance. However, they did not consider the multiple path problem or algorithms that do not optimize the cost metric such as Multi-Constrained Path (MCP). In our case, most of the routing is based on the shortest path algorithm. They differ in the way they treat the traffic and which optimization QoS metrics they chose to make the routing decision. Moreover, many authors argue that these metrics depend on the application on which routing algorithm will serve which is not the case in real life networks where diverse application coexists.

Table 7 shows a summary of reviewed routing works been done in SDN for QoS provisioning. We did not show the shortest path problems due to space limitation.

2) RESOURCE ALLOCATION

It is necessary to manage network resources in a way that maximizes its utilization. Competition between services on those limited resources complicates such tasks. SDN shows potential to resolve such issue with its dynamic programmability which allows network owner adapts to new requirements efficiently. In this Section, we discuss various approaches in the literature to cope with the resource allocation problem.

a: RESOURCES RE-CONFIGURATION

The work in [219] utilized weighted metric to compute the utilization of end-to-end paths (ingress-egress) and use this metric for calculating routes for QoS traffic. The exciting part is that when the bandwidth requirement for the QoS cannot be achieved, a resource evacuation is performed. The approach starts gradually removing the best effort flows from that path to allocate extra bandwidth for the QoS flow(s). It starts with a single best effort flow, if it was not enough multiple flows are rerouted. For resource allocation, the controller configures outgoing ports along the chosen path to handle the QoS flow requirements (e.g., max or min rates). The controller uses OF-Config protocol [75] for this task since it is beyond the scope of the OpenFlow protocol. Authors assume the existence of northbound APIs for users QoS requests, however, the QoS flows are predefined within a file that is checked by the controller dynamically.

The authors in [245] propose a traffic management using SDN. The controller monitors the behavior of currently flowing traffic. When it detects a degradation of the network performance or new demands is received, it first reconfigures network devices to reflect a ratio-based bandwidth allocation. If the available bandwidth cannot meet the new requirements, a low priority flows are evacuated to another path(s) and use their share for high priority flows. However, the authors did not clarify how they configure the bandwidth parameters since it is not supported by OpenFlow yet.

Similarly, the work in [239] uses link utilization as a cost metric for path selection for QoS flows. Whenever a routing decision cannot meet QoS requirement, queuing techniques are used to enforce prioritization of the QoS flow to meet its requirements. They reserve bandwidth on queues along the path the QoS flow will use. Authors ignore the overhead of monitoring the status of link utilization when they are polling statistic from the network element. This issue is vital since the accuracy of the network affects routing decisions and it is a trade-off between overhead and the accuracy (i.e., actually they did not show how statistics is polled).

In [240], the authors proposed an approach for network resource allocation based on different requirements of multiple services or applications. Their design consists of four modules: a traffic classification at the edge of the network (i.e., DPI-based software for getting flow information and send it to the controller), a queuing module, status collection (i.e., utilization of queues), and path calculation. After the controller knows which service group a flow belongs to (i.e., based on the information received from the DPI classifier), the queuing module configures queues within the network switches to support specific requirements for each service (i.e., bandwidth and delay are the QoS metrics). Therefore, the controller maps each flow to the queue(s) that will satisfy its requirement depending on the service type that flow belongs. For path calculation, they modeled that problem as constrained shortest path (CSP) and used the LARAC algorithm to solve it. They optimize the path for each group of services based on their most important QoS metric (i.e., VoIP sensitive to delay). Classification at the network edge is like what is used in today networks. However, usage of deep packet inspection may consume more resources at the edge switch. Moreover, instead of tagging the packets, they send the information to the controller.

In [216], the authors leverage SDN to support resources allocation for cloud users. They utilized QoS route optimization combined with queuing to ensure resource allocation for high priority users. They divided the flows into QoS, and best of effort flows using ToS field in the IP header. Both flow types can share the same path(s) when the network is not congested. They poll statistic from network switched to monitor the network state. Optimized LARAC-based routing is used to compute a path for QoS flows based on their requirements. Where there is no feasible path found, they do queues manipulation through the OVSDB [262] protocol to enforce the resource allocation. Authors ignored the communication overhead between the controller and switches while statistics is collected or when rerouting is needed.

The authors in [241] propose SDN-based real-time dynamic traffic shaping and bandwidth allocation for clients in home networks. When a client requests a video streaming, controller intercepts the DASH video streaming communication to get meta-data such as the video length and the bit-rate. Then, a QoE optimizer module tries to find the optimal allocation of bandwidth between clients to maximize the bit-rate each receive and equitably share the bandwidth between client as it is specified by network policy (i.e., they use equal bandwidth). Based on that, a traffic shaping module

TABLE 7. QoS routing in SDN.

Ref	Objective	Constraint	Formulation	QoS traffic	Controller	Domain	Network	Control	SDN Plane
[213]	Minimize a weighted sum of the total route length for QoS and the packet loss for other traffic	No QoS traffic loss with tolerable path length/delay	Constrained Shortest Path (CSP)	SVC video base layer	NOX	Single	Campus	Single	Control
[214]	Minimize weighted sum of packet loss measure and delay variation	delay variation	CSP with LARAC	SVC Video (base and enhancement layers)	–	Multi	–	Single	Control
[215]	Minimize weighted sum of the route length of rerouted base layer traffic and packet loss rate	path length	CSP	Video base and enhancement layers	LEMON Simulation	Single	–	Single	Control
[216]	Minimize weighted sum of packet loss measure and delay variation	delay variations	CSP with LARAC	Video streaming	RYU	Single/ Multi	Cloud	Single	Control
[218]	Minimize sum of congestion and delay measurements of path links	delay variations	CSP with LARAC	Video streaming	Floodlight	Single	–	Single	Control
[212]	Minimize weighted sum of packet loss measure and delay	delay	Delay-Constrained Least-Cost (DCLC)	Video conferencing	Floodlight	Single	Campus	Single	Application
[196]	Minimize video distortion	delay	Multi-path routing, sub-gradient-based algorithm	Video streaming	–	Single	Enterprise	Single	Control
[219]	Minimize the delay	bandwidth	-	Video streaming	POX	Single	Campus	Single	Control
[220], [221]	wighted cost of delay and loss $= (1-B)*d+B*p$	delay	CSP solved with LARAC	SVC base layer	LEMON Simulation	Multi	–	Single/ Multiple	Control
[222]	sum of packet loss ratio on the route, or the sum of link's packet loss ratio	delay	CSP solved with LARAC	base then enhancement layer for SVC streaming	OpenDaylight	Single	Enterprise	Single	Control
[223]	Minimize delay	delay variation	CSP solved using LARAC algorithm	video stream :base (1-level QoS), enhancement (2-level QoS)	Floodlight	Single	Campus	Single	Control
[186]	Minimize the transmission delay	bandwidth and max (H) hops	CSP	assumed to be delay sensitive services	RYU	Single	Enterprise	Single	Control
[224]	per-hop cost or buffer-aware cost functions are tested not both	delay	DCLP	–	Simulation	Single	Campus	Single	Control
[225]		delay	DCLC solve as kLAM approximation	–	Simulation	Single	Campus	Single	Control
[192]	Aggregating reliability	Reliability, bandwidth, delay	Multiple constrained shortest path, A* Prune algorithm	Video streaming	–	Single	Campus	Single	Control
[228]	maximize the sum of MOS values over all multimedia sessions	receives a minimum MOS value, Maximum Link rate is conserved	MCSP solved as Integer linear implemented IBM Optimization Programming language	Video (e.g IPTV), Voice(e.g VoIP), Data (e.g. FTP	Simulation	Single	Campus	Single	Application/ Control
[229]	Wighted sum of the satisfaction ratios of each required demands. The weight is adjusted by the miss rate of that QoS demand	delay, loss, bandwidth	MCP try to estimate with simulated annealing, T=16 is the max iteration	Assumed to be just QoS flows with demands	Floodlight	Single	–	Single	Control

enforces the output of the optimizer into rules installed by the controller at the home network gateway.

In [242], the authors used OpenFlow meter table for multimedia bandwidth assurance. They used the meter table entries to do metering for flows, while traffic is classified based on the incoming ports and the IP destinations. If link's congestion is detected a rerouting is performed to relieve that link. The controller periodically collects statistic from the SDN switches. They predict a bandwidth value for each flow and compare it with the corresponding real-time bit-rate reported from the SDN switch. If the predicted value less than the real value, a multiplicative increase is performed, else an additive decrease is performed to release part of the reserved bandwidth. It looks likes the congestion avoidance mechanism used by TCP (AIMD) but reversed. Links âĂŸcongestion can be detected from collected statistics. Authors used a rerouting algorithm that distributes low priority flows traffic along others link. The number of the packet forwarded to each of the new routes depends on the amount of available bandwidth that link has. They used OpenFlow group table to perform this multi-pathing. This work depends mainly on two things, the detection of congestion in the network which depends heavily on statistic collection which means higher accuracy requires a higher rate of statistic packets either pushed or polled from switches which may add extra overhead. The other thing is the prediction equation used for bandwidth reservation. From the results, they showed that they predict the peak traffic, but between peaks, degradation is slow which waste resources reserved for multimedia traffic.

The authors in [243] derived a model for evaluating the performance of SDN architecture. They also proposed the usage of two queues; one has a high priority for packets from the SDN controller, and another low priority queue for other packets. The purpose of the high priority queue is to reduce the delay when flow miss-match happen for packets. They provide higher priority for those packets coming from the SDN controller. Their purpose was to maximize the fairness between packets with little scarification of performance (packet loss). They evaluated their approach using mathematical model and simulations. The results show lower delay for packets that suffer from flow miss match better than the traditional FIFO scheduling. This work is like what existing in current traditional network elements by providing high priority for important packets such as VoIP. Therefore, the authors did not change much but used this feature to help early flow's packets that suffer delay when a decision from the controller is required.

The authors in [244] propose a queue migration scheme. They used a model for predicting the delay of queues. If the queue delay is high and expected to violate the deadline constraint, packets are migrated to different queues.

b: NETWORK VIRTUALIZATION

Another way to provide and assure resources allocation for users is by dividing the physical network into virtual slices or networks. Isolated from each other with resources guarantees. A virtualization layer resides between the control plane and the data planes. *FlowVisor* [253] is one of the earliest works that does network virtualization by creating a virtual layer on top of physical network devices. Separated slices or virtual circuits can be created using *FlowVisor* where each can be thought of as a set of flows that share common features. *FlowVisor* is built based on OpenFlow SDN architecture and works as a proxy between the control plane and data plane. For instance, source and destination MAC or IP addresses can be used to define a slice with specific QoS requirements such as maximum delay or jitter. These features called flow spaces and *FlowVisor* ensures isolation between them. However, it does not allow sharing of port bandwidth between multiple flows (i.e., the port for one slice). Also, it does not support manipulation of packets. Other proxy-based virtualization works are proposed later on to overcome some of the limitation in *FlowVisor* as it is the case in *OpenVirteX* [254] which built based on *FlowVisor* to tackle the address virtualization problem. *DFVisor* [255], [256] attempts to enhance the flow setup time since flow setup process is intercepted by the hypervisor to make each controller think it controls the network. *AutoSlice* [257] proposes schemes to automate the virtual slice deployment and solve the single point of failure in a single proxy based virtualization by distributing that task which makes it scalable for a large number of users.

Similar to proxy-based virtualization, the work in [258] proposed a fine-grained bandwidth allocation manager (FGBAM) that is consist of two module; web-based front-end for bandwidth allocation policy specifications, store it at the back-end manager. *FGBAM* enforce those policies by acting as a proxy between the controller and network switches by intercepting OpenFlow messages. It allows port bandwidth among flows which is not the case in *FlowVisor*. Also, *FGBAM* can work with different network operating systems along with different SDN deployments since it uses OVSDB protocol [262] to manage bandwidth at network switches instead of OpenFlow.

Based on *FlowVisor* slicing, the authors in [259] designed a traffic shunt system. *FlowVisor* is used to divide the physical network to different virtual logical network or slices. The VLAN tag is used to identify network slices in which different traffic sources are steered to that slices. A bandwidth allocation module is used to allocate bandwidth on network slices as needed. By doing that, bandwidth starving application or traffic can be diverted to network slices with fixed bandwidth. Also, the authors in [260] present a hierarchical link sharing and bandwidth reservation in SDN-based virtual networks. They use *FlowVisor* to create multi-level virtual networks in which a virtual link can span multiple physical links.

For resource allocation within the network using virtual circuits, the authors in [247] proposed a QoS provisioning architecture. It uses two variant of virtual circuits (VC) creation within the SDN network. The first one is fine-grained in which the QoS profile is applied for each VC even if many circuits share the same physical path. Another coarse-grained is investigated, in which multiple VCs share the same QoS profile. Within the QoS profile, service classes (e.g., ToS) is used to differentiate between virtual circuits. Creation and manipulation of QoS profile are done through the OVSDB protocol and realized through priority-based queuing and rate limiters to meet the QoS requirement for each flow(s).

In [263], the authors did network slicing and provisioning for these slices to meet application requirements the controller reserve resources for customers based on high-level slices specification defined by the administrator for customers or services. The slices can be applied on aggregated or granular flows. They proposed schemes to solve the problem of meeting high-level requirements with low-level configurations. They configure rate limiter at network edge (i.e., bandwidth usage policies) and priority queues at network nodes (i.e., for bandwidth and delay reservation). This procedure is similarly done in current network architecture, but manually. They leverage SDN controller to automate these tasks. They still do manual configuration of the application flows classification. Therefore, a dynamic detection (e.g., application profiling) will improve this work and reduce errors.

The authors in [261] proposed a management framework that joint off-line network virtualization and QoS aware routing. First, the physical network is sliced into different subnets to isolate different tenants. For each tenant, a subgraph is defined in which it contains all vertices in the original graph G. Resources are isolated, and the QoS is guaranteed for each tenant or client. When new flow(s) arrive, a routing algorithm handle the flow allocation that uses client properties to realize their QoS requirements. When routing fails in accommodating QoS requirements, feedback is issued to the off-line stage to enable better network slicing.

c: OPTIMIZATION

The authors in [264]formulated the online virtual links resource allocation as an Integer Linear Problem. In which requests for virtual links arrive sequentially to the system with specific requirements such as delay and bandwidth. The objective of the system is to distribute the network traffic with appropriate network resources utilization taken into consideration. This behavior can be applied to resource allocation requirements for flows in which single or multiple flows may produce single or multiple virtual link request with those QoS requirements. The output of the algorithm is a route(s) and set of flow table rules that should be installed on the physical nodes or switches to support the creation of virtual link(s). They argue that the distribution of virtual links creation among network node increase the admission rate because nodes and links are reasonably utilized.

Similarly, the work in [265] formulated the problem of virtual link creation on top of physical network as a Multi-objective Integer Programming (MIP). However, each virtual link request is associated with a QoS class that is defined within the QoS profile (i.e., specific attributes with absolute values). They applied three general link sharing policies when creating a virtual link over the physical link. Full sharing policy allows the physical link to be shared with as many virtual link requests with no constraint on the QoS class that request belongs to unless the max link capacity is reached. The second policy is called *full split* in which each the QoS class can have a maximum share of the physical link. It is for the equitable utilization of the physical resource between different QoS class. In this policy, different QoS class can share the physical link. The last one called the Russian Dolls Model in which the physical link is shared in a hierarchal way between the different QoS classes. This policy support priority between classes in bandwidth sharing which like ToS definition with lower values means higher priority. This division of QoS class and its constraints on creating virtual links can be considered such as allocating specific virtual links for certain types of traffic to satisfy its requirement. It is necessary due to the burst nature or unpredicted behavior traffic of those services or applications.

The authors in [248] modified the NUM (Network Utility Maximization) framework to support multi-class services. The objective of the NUM framework is to maximize the sum of the network flows utilities within the limitation of links capacities. They used two different utility functions for both elastic and inelastic flows. Each flow's utility function has a priority parameter to reflect the QoS class for that flow. For a new flow, and after routing path is computed, the controller uses a sub-gradient projection algorithm to estimate or allocate the rate of that flow. The only considered constraint here is the link capacity. For scalability purpose, the authors assume usage of multiple controllers. They used *Opnet* to evaluate their work.

The authors in [249], [250] leverage traffic prediction for QoS-aware resource reallocation. They proposed ways to minimize the packet loss ratio while maintaining the delay and the bandwidth under control. They model the problem as Binary Linear Programming and propose two schemes to solve it. An exact optimal solution (QRTP) which solve the problem using CVX toolbox in Matlab. However, to reduce the optimization time, the made two relaxations on the original solution. The relaxed scheme is called *RQRTP*. First, an aggregation of flows is performed. Instead of granular-based flow optimization, they join flows that share the same source, destination and have a total bandwidth less than a certain threshold (i.e., forwarding table compression). Then, an upper bound for the objective function is computed to minimize the total link utilization. The approach computes the link utilization as the sum of the flows traversing that link. The resource reallocation process is ignited only when congestion is detected/predicted, or the time for the periodic resources optimization is reached.

d: SCHEDULING/ADMISSION CONTROL

The authors in [251], [252] introduced a different approach for resource allocation targeting fairness. They exploited the principles of virtual memory management in computer and applied it on SDN platform. For each control application, they maintain a set of state information or tables within the control plane. Each control application contains three tables contains information about its created flows. The first table contains information about flows that need to be scheduled and written to switch's flow table named as Virtual Forwarding Page (VFP). The second one keeps statistics about those flows belongs to a particular application named as Flow Performance Table (FPT). FPT is used later to compare achieved bandwidth reservation for that application. The third table called Flow Status Table (FST), and used to keep the status of flows that are currently or lately active within the network. A binary value (e.g., 0 or 1) is assigned to each flow within that table based on their activity within a specific period. This value is used to determine if the flow is recently used or not. The overall application throughput or the total allocated bandwidth is computed based on the collected flow statistic table (FPT) and the activity of those flows belonging to that application (i.e., FST table). New application's flow is scheduled if the max throughput for that application is not reached yet. The flow is inserted into the flow table within the switch if the table is not full. Otherwise, the status table, FST, is checked for those flows that are not used recently to evacuate and replace with the newly arrived flow. In this work, the SDN controller needs to keep track of much information about each flow and each application. Moreover, they just used bandwidth as a resource requirement. However, if multiple application requirements are considered such as delay or jitter, the scheduling may differ.

e: RESOURCE SHARING (AUCTION/PRICING)

The authors in [266] exploited *FlowVisor* and virtual network slicing to present an auction-based resource allocation. Authors assumed exitance of multiple controllers with no interaction between them. Each controller managed and control a slice or a virtual topology defined by *FlowVisor*. Therefore, flows belonging to a specific controller' user(s) follows a certain set of virtual links. The problem they address is that controller may need extra resources to support incoming or existing flows of its users. *FlowVisor* performs the role of auctioneer, and interesting controllers behave like bidders. Due to its knowledge of the status of the network and existing network slices, *FlowVisor* behave as a proxy for all controllers in which it receives bids from each one and release shared resources to an interested controller. However, each controller will pay the cost to get those resources and the *FlowVisor* makes the auction decision. Authors here assumed the existence of multiple controllers, and even it may be not always applicable. However, this approach can be used to allocate resource for users sharing the network resource. Each user wants extra resources must pay the extra cost, and it may get those resources from a similar user that has underutilized private/reserved resources.

In summary, we can make the following remarks:

- In routing, the cost function plays a major role in finding a route that statistics the QoS requirement
- Beside that, accurate routing is affected by the accuracy of the maintained network state. Therefore, a well-engineered monitoring function will have functional consequences on other modules such in routing.
- Multi-domain routing is another issue, especially when considering QoS provisioning. In [220], the authors generalized their local domain LARAC-based QoS routing to address the distributed multi-domain problem. The path between two domain border nodes is modeled as a virtual link. Therefore, the link cost and delay are computed according to two methods: 1- the cost of the virtual link is the total cost of the path between border nodes, and so on about it is the delay (delay of the path). 2- find k feasible paths, by solving CSP multiple times, and then compute the average cost and delay of them, and assign it to the virtual link.
- Another thing needs to be considered, is that most of the works focus on a specific application and optimize routing. The authors in [233]–[235] generalized the optimization and provide an API for solving constrained programming problems. They proposed a platform for facilitating optimization for the user, by making interface base on constraint programming. The user specifies the constraints, and the solution, if found, provided by the framework through another program called the *solver*. The point here is such work can be used simultaneously for different applications to accommodate various requirements.
- Usage of AI or more specifically ML techniques can help to achieve more fast routing especially if the models are well trained. However, few efforts were performed compared to mathematical optimization model. Even it has a long training time, it can give fast result if the right model is produced.
- OpenFlow has limitation in for configuring resources, and controller depend on other switch-specific configuration protocols such as OVSDB. OF-Config protocol introduces to alleviate this problem.
- Network slicing for resource provisioning is suffering from the isolation problem. Same resource cannot be shared with more than one entity as it is the case in FlowVisor.
- Application of ML techniques in resource usage optimization is not considered. Learning models taking the time dimension with utilization history usage can help in optimizing resource usage in conjunction with predicted traffic models.
- Resource sharing between entities need more attention, specially when allocated resources is less utilized by the tenants.

D. EXECUTE

The execute function is responsible for reflecting a change requirement into the network behavior. Since SDN controller do (re)configuration of the network in behave of upper application. This process needs to be performed carefully due to its direct effect on the network state, and the whole SDN idea. SDN controller controls the network by sending flow modification command to OpenFlow compatible switches (i.e., OpenFlow agents reside on those switches) to install flow rules within forwarding tables [267]. Based on those rules, any policy (i.e., hypothetically) could be applied to the network either for security, QoS policies and so on. From the data plane point of view, it expects two types of plans from the controller to cope with changes in QoS requirements. The first one is a request(s) to modify one or more entries in the flow, group or meter tables. New flow rules may be added/deleted to reflect the addition/deletion of flow to/from a specific path. The second type is a request to add/remove resources to specific existing flows.

The consistency of network configuration is a significant issue in the design of SDN-based autonomic networking. Manipulation of flow rules changes the network behavior. Therefore, it is necessary to verify those rules and their consistency of what already implemented by previously installed rules. Moreover, the timing of those update may affect the ongoing traffic and the overall network performance. Therefore, scheduling of such updates should be performed carefully. Accordingly, the essential question of when and where rules to be installed need an answer.

1) RULES SCHEDULING

The effect of flow rules insertion may not be limited to the violation of implemented policies. The authors in [269] showed that the sequence these rules are inserted can affect the network performance. They suggested a scheduling algorithm that runs at the SDN controller which firstly assesses the effect new will be inserted rule on current network and if it can cause congestion for network links. This problem may become obvious when a rerouting decision is taken by the routing module without considering the effect of flow migration or the sequence of flow entries changes.

An update to the network could be a flow rules modification or deletion due to a change in policies or requirement of traffic engineering application. Which will changes packet processing behavior within the network. The authors in [270], [271] build their work based on the observation of the existence of interdependency within network updates. Such updates may contain sub-updates that are independent in which they can be installed in parallel within the network. Sub-updates are independent, for instance, when they are targeting different switches or network slices. Therefore, the authors build a scheduler called *ESPRES* that runs as a layer between the controller and switches. It receives a stream of updates to schedule their installation without overloading network switches.

Instead of pro-actively install many rules within switches (i.e. flow table size is a big issue due to memory limitation) or reactively consume network resource by gradually inserting flow rules, the authors in [272] proposed a hybrid way, in which they divided flow rules installation process into path and node based. Path-based rules early installed on edge switched then gradually along other network switches.

Control-Data plane consistency is outside the scope of this work. However, for its importance, especially when multiple controllers deployed in a single network [273], we mentioned some of the effort put in that area. The reader can find in [274] a recently published survey focused on update consistency in SDN.

2) QUEUE MANAGEMENT EXTENSION

OpenFlow is mainly designed to control the network behavior with a simple manipulation of the flow entries (i.e., add/modify/delete). Even the subsequent versions of OpenFlow add more features such as the ability to direct a flow(s) to a specific queue(s) or metering those flows (i.e., as discussed in SectionIII-B). However, it cannot configure the network resources (e.g., ports or queues on the switch). The ONF group delegates the port/queue configuration to another protocol called *OF-Config* [75].

Therefore, some effort was performed to fill that gap in the literature. For instance, in [275], the authors tried to apply traditional QoS techniques; traffic shaping and *t*o SDN-based networks. They developed a QoS module within the Floodlight controller to do that. That module performs flow classification, processing and policy application to OpenvSwitches. Flows are directed to previously configured queues by using the OpenFlow *enqueue* action. They manually configured the queues by utilizing existing OpenvSwitch management protocol, the OVSDB [262]. This due to the limitation of OpenFlow and the *OF-Config* protocol was still under development. Similarly, the authors in [277] developed a QoS module in the Floodlight controller. It uses OVSDB commands to (re)configure queues bandwidth reservations along the path nodes between the applications and their replicas in the cloud. It helps in high availability guarantee of such applications to their user.

The authors in [276] developed a Floodlight module called *QueuePusher*. That module enables the controller to perform queue management in OpenFlow enabled switches. It translates queue management messages into OpenVSwitch compatible commands through the OVSDB protocols. The module offers certain operations such as CREATE, READ, UPDATE, or DELETE (CRUD) to manage. *QueuePusher* is presented as an extension to the Floodlight protocol since does not support OVSDB. Moreover, it provides a RESTul API for this module to be instantiated from top layer application or other entities within the controller. We need to mention here that the IETF group standardizes the OVSDB protocol. *QueuePusher* is also used to configure queues for flows QoS support in [278].

Similarly, authors in [279] built an SDN plug-in or an API to configure OpenFlow switches queues based on the OVSDB protocol. Similar to *QueuePusher*, it will help in dynamically configuring QoS requirement such as the max-min rates, burst size and priority of the queues on these devices. Authors just did what the OF-Config protocol should do, by translating management commands into internal OpenvSwitch management command. The exciting part of their work is

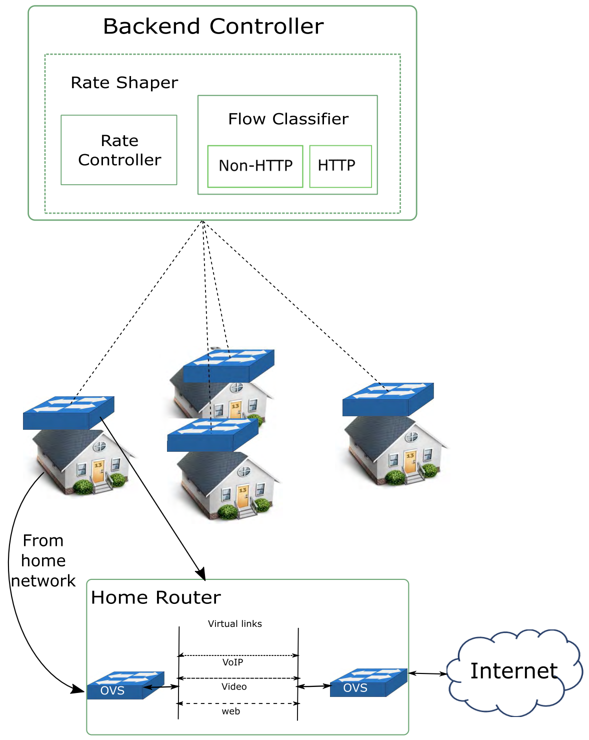


FIGURE 13. Illustrative diagram for *FlowQoS* scheme as reported in [281].

providing an API for top layer management application to configure the low layer data plane.

The authors in [280] tried to benefit from existing traffic management, or packet scheduling techniques exist in Linux. Such as the HTB for traffic shaping, RED to avoid congestion and SFQ for fair classless scheduling. They aimed to perform such queuing in OpenFlow switches since in OpenFlow 1.0 (i.e., the supported stable version by that time) assumes FIFO and rate-limiting as the only QoS support. Therefore, they added a QoS module in the OpenFlow data-path to provide OpenFlow QoS messages that abstract the queue configuration (i.e., the traffic shaping, packet scheduling). That module issue commands to install any of these scheduling algorithms to port queues by utilizing the underlying Linux kernel. It uses the TC Linux tool for schedulers installation.

The authors in [246] extend both OpenFlow and switch hardware by implementing a NetFPGA-based OpenFlow switch. It allows the SDN controller to configure quota values specified for each queue that uses Deficit-Round-Robin (DRR) scheduler. The value of quota specifies the amount of bandwidth that the queue receives.

The authors in [281] use SDN-based home or broadband traffic (i.e., user's HTTP based video traffic) management. There developed two modules, a traffic classifier, and rate shaper. They used DNS information to classify web applications such as HTTP(s) or any built upon it, and another classifier that depends on deep packet inspector to classify non-HTTP traffic. They used two internal virtual switches within the home router to provide rate control for different flows. Virtual links between those switches perform the traffic shaping. Therefore, after a flow is classified, a rule is created by the controller to direct this flow to one of these virtual links that correspond to an application group such as video, gaming or web as depicted in Fig. 13. The traffic shaping applied user-defined rates on these links. They monitor these links and reconfigure the rate assignment to maximize the utilization of the link capacity. This approach requires two virtual switches within the manage gateway which is a pressure on the device resources. Similarly, authors [282] used SDN-based home routers configuration to provide traffic prioritization or allocation of bandwidth. The controller uses information send from end hosts to tell which applications (i.e., Netflix or Skype) are running by the users right now (i.e., classification). Then, the controller computes and optimally share the bandwidth between these applications based on a predefined utility function for each application. The proposed approach delegates the traffic classification task to agents installed on the user's host devices. Similar to the *FlowQoS* [281], they used two internal virtual switches with virtual links in between to apply rate control.

In summary, we can make the following remarks:

- Due to limitation of OpenFlow specification, some research efforts were exerted to provide interface that translate from application-level into switch-level configuration.
- Policy conflict and consistency between the data-control planes is a major issue. Many applications could program the network. Therefore, a central view and automated control of all policies need to be implemented.

E. KNOWLEDGE

We previously discussed the main four functions in the autonomic MAPE-K QoS management. The remaining one is the Knowledge which we discuss in this section. We review the storage of all type information either received or produced by the QoS approaches. The network state information, configuration or policies controlling the operation of the system are stored in knowledge source.

Most of the existing SDN controllers hold and maintains information about the network such as existing devices or network topology. For instance, in the Floodlight controller, a device manager module maintains information about discovered devices or host (e.g., from *PacketIn* messages). Link discovery module uses LLDP packets to discover links between switches, and such information is used to build and maintain the network topology by the topology service module. For information sharing between modules, an in-memory storage source is used in Floodlight. Modules can manipulate (i.e., create/modify/delete) such information and kept updated when changes occur.

Most of the previously researches in the monitor function (i.e. Section IV-A) store collected measurements locally. For instance in [283] uses a persistent repository to store infrequently network information such as the network topology, exiting paths. They store monitoring requirement received from the upper application in tables that will be processed by the monitoring unit. A collected statistic such as link utilization is stored in hash-tables. The authors in [107] use database module to store the flow counters collected by the monitoring module. History of such measurements is used in their approach to predict future flow(s) rate. Similarly, in [109] raw measurements are stored and maintained by a data store module. Approaches in [86], [185] uses local database to store up-to-date statistics and topology information. The approach in [123] keeps statistics information in its database. In every 100ms it produces a snapshot database for other application. Such database contains information about, topology, statistics, flows classification (i.e., five tuple information), flows paths and their estimated bandwidth. In [284] MongoDB [285] is used as database service. It is used they store information received from the controller messages such as the *FlowMod, FeatureReply* (i.e. switches information). Also topology information captured from the LLDP packets. All together with collected statistics are used to visualize the current state network. Differently, the authors in [131] store history of control messages sent and received between the controller and network switches for logging similar purposes. In their used storage or database entries a per-switch (i.e. using DPID of the switch as identifier) is created. A GUI is created to access and visualize such data to the network operator.

The authors in [286] used MySQL DBMS to store network, application and user information. Such as the topology information, sent/received packets, QoE parameters received from the user side (i.e., after watching the video), QoE threshold, bandwidth usage. Then, QoE value computed and compared with specified thresholds. If low QoE is detected, the controller tries to improve the QoE by finding better paths. Moreover, user bandwidth is monitored in which if it exceeded purchase bandwidth, admin is notified.

The authors in [287] proposed a recommendation framework to mitigate detected elephant flows in IXP(Internet Exchange Point). It uses a set of predefined templates IXP operator. Each template contains a procedure to handle the detected elephant flows. For instance, a template could be to find the best alternative paths for the current detected flows. The operator have the choice which templates to be activated based on their needs. The active template is applied on all lastly detect elephant flows. Based on the operator requirement, the recommendation generates recommendations after applying a specific template either on all or subset of the detected flows. Recommendation takes the of flow table rules modifications. The operator validate such recommended rules and either approve to be applied or request a change. The operator can modify template through template manger. He/she can also more templates to be used in the future for other objective. This approach can be used to provide autonomic mitigation of elephant flow. Experience of engineers can be converted to such templates to handle different situations autonomically. However, in their approach, the operator interfere the automated process.

The authors in [147], [288] propose a policy-based management system. They assume that SLOs is already defined and verified from the SLA, for instance by the network operator. The system then stores the QoS SLO policies in an internal database or repository. Their proposed approach tries to reflect such policies into network policies. When a policy violation is detected, the policy enforcement module takes one of two actions: either rerouting the best of effort traffic and give high priority to QoS traffic. The other action is limiting the rate (i.e., meter band that drops best effort packets) of the best effort traffic that share the same path with the QoS traffic.

Similarly, the authors in [156], [157] define SLA and SLO repositories. They are maintained and edited by the operator to reflect new changes or requirements. Moreover, they designed a smart SLA enforcement engine, in which, an SLA/SLO violation are predicted. A forecasting unit tries to predict the future traffic. Then, an ANN based SLO violation prediction is performed. Therefore, such situation can be handled before it occurs which the role of the enforcement anent in the smart engine. Actions can be issued to the network such as increasing the resources for certain users or application.

A network management engine is developed in [268], called *GolfEngine*. Fig. 14 shows the design of the *GolfEngine* system. It consists of three layers: User interface, Database, and Runtime Manager. The Runtime Manager uses stored policies to control top layer application behavior. The system uses MySQL as the DBMS to store variant type of information: 1-Application policies: policies that control the application behavior to define its profile such as the polling policies (i.e., for statistic collection), analysis policies (i.e., threshold-based comparison with what the in polling policy), and an action policy (i.e., defines how the application can perform actions such as drop/deny). 2- Network Flows Information: stores statistics polled by the controller updated periodically. 3-User Information: logs the user behavior on the system. Policy-based application control is similar to what autonomic networking is pursuing. Policies are defined by the system operator to make them self-controlled.

In summary, we can make the following remarks:

- Policy-based management that involves SLA enforcement and smart violation prediction engines that get benefit of existing machine learning techniques is a major improvement toward the autonomic behavior of the network or more specifically the QoS management in our case. However, few works that implement such features. Moreover, policy-based control of other MAPE function is limited in the literature. Most assume specific scenario to just improve QoS/QoE performance without considering the automation of such management tasks.
- In most of the cases, policies are operator defined. There is a limitation of machine produces policies that can use history information to infer new policies that may improve the system performance. Especially consider

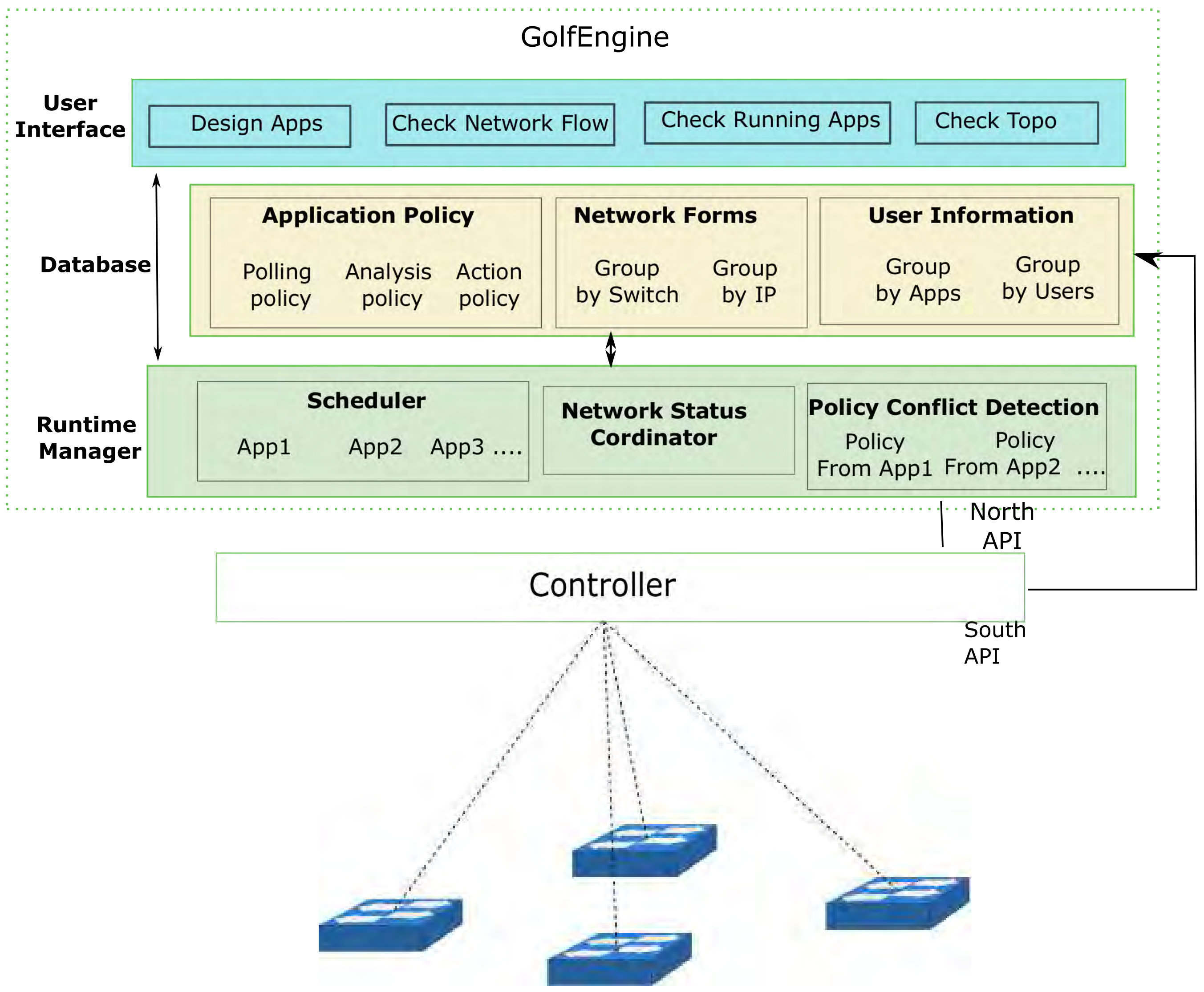


FIGURE 14. GolFEngine architectural model as reported in [268].

the time dimension to improve the daily action and plans for network operation.

V. QoS REQUIREMENTS SPECIFICATION

In this section, we review research efforts in which authors identified user QoS requirements for the SDN controller which translated into network configuration. Negotiation of the requested services between end users, service provider and the network operator take many forms, and as a result, they should agree on an SLA. For users or business owners, it essential to guarantee the quality of the requested service. However, due to the dynamic changes of users and business needs, renegotiating terms of services is a time-consuming task. Therefore SLA may be applied for an extended period.

Another issue is the reconfiguration complexity of network elements in the current network architecture to adjust QoS policies for a short time interval and roll it back to normal state after services time. Which reflect the major need for transferring to SDN networks. Through northbound interfaces, network application can specify networks requirement to the controller to create flows rules. Whenever there are no more needs, the SDN controller can delete and install new rules to roll back the network to its previous state. For instance, a user wants to perform a conference call to another one or attend an online course. The user application will negotiate service and network communication parameters between end users before starting the flow of the service (e.g., audio or video). The network SDN controller should understand that to converts these parameters into flows' rules within network elements. Therefore, there is a high demand for automating this task, especially for short period flows.

In [236], the authors used SDN for resource management within the network. They evaluate the performance of YouTube video streaming using pre-buffered play time as the QoE metric. A higher value of this metric means low stall time for the video stream which a good indicator of customer satisfaction about the ongoing service. Authors propose to allow the streaming application to contact the SDN controller through north-bound API to tell about the type of information its flow contains. It allows the controller to put a high priority of such flows over other existing flows. As expected, the minimum threshold they put for adequate pre-buffering time for video streaming is reached due to the high priority YouTube stream flows receives over other TCP flows. This work is a case study of allowing the application to help the network providing good QoS. However, this work depends meanly on trust by the SDN controller of the information provided by the user's application (i.e., they assume to use application signature). Also, there is no competition between multiple flows of the same data type or other critical flows or different users. Another issue is that there should be a clear identification of granular flows and providing a standard north-bound API with a clear definition of the information to be exchanged between the two entities.

Similarly, the authors in [237] used SDN to provide network resources management with the benefit of application information. They used YouTube video streaming as the case study and the buffer playtime as the application information that fed to the SDN controller. When the buffer play time of flow is below a certain threshold, resources are reserved for that flow; otherwise, flows use the same amount of resources. This work like the previous one in the way they involve application awareness in the network configuration. They also used a minimal access network to test their approach which needs more investigation in large scale with many competing applications.

The authors in [289] proposed an SDN-based architecture for service negotiation. The architecture uses, besides the controller, a QoS module to match the QoS requirements or the ability between the end users. After negotiation, an optimized QoS profile is produced for the expected flow. This module called QoS Matching and Optimization Function (QMOF). It uses the SIP protocol for signaling, and then a path optimization is requested from the control layer for this flow through the Path Assignment Function (PAF) which reside on the controller. After that, a path is configured along the network elements, and the flow is started after returning an OK signal to the QMOF module after configuring the flow entries. Then the SIP signaling is completed between the two entities and the flow start. This form of service negotiation will not consider the state of the network in traditional architecture, since only end entities are involved, however, usage of SDN controller allows the network to produce a profile of the service for the end users for getting the best service QoS without congesting the network.

SLA application between entities consists of two parts. The first is the specification of service or network parameters that are important for the quality of the service to be within predefined ranges. The second is the monitoring of these parameters (e.g., maybe a third party to ensure no violation) and may adjust them to satisfy the customer. In [290] authors proposed PolicyCop, an architecture for automatic QoS enforcement on SDN. The architecture consists of two modules that reside above the control layer. The first component is specialized for detecting any violation of current policies (i.e., monitoring the traffic and policy database). If a violation is detected, action request either to the manager to handle manually or to automotive policy adaption or enforcement module which is the heart of QoS automation PolicyCop that represent the second component. Policy adaption component is containing a set of policy adaption actions. There is an action for each policy violation (e.g., a violation in latency metric or throughout). This adaption or enforcement of policies reflected in resource provisioning requests to the control plane. These two modules communicate with the underlying control layer using the northbound API. The PolicyCop exploited some existing modules within the control plane; namely: admission control (accept or reject resource provisioning), a routing module, device tracker, Statistics Collector and Rule database. From the SLA point of view, the specification of SLA is represented by the current policy database and the monitoring of these specifications or objective are done by the PolicyCop which may enforce action in the network exploiting the programmability of the SDN. The detection of violation of the QoS SLA may require a long time in traditional network especially of reconfiguring the network to adapt of policies, however, this task can be automated with the appropriate API with the SDN controller to allocate required resources. It would be more valuable to this work if there were an automatic translation of SLA objective or specification into policies since authors assume the existence of policies within a predefined database. It may not be applicable, but dynamic changes to the SLA should be reflected automatically to network element upon the agreement on these changes or if the customer pays for these extra enhancements. This work is an example of QoS provisioning based on policies that autonomically adapt to changes and configure itself when SLA violation is detected.

Customer QoS requirements may be specified as SLA, and changes are always expected. SLA is high level (QoS user requirement) policies that should be translated into low level (system policy) network policies by network administrators. Refinement of higher polices is an exhaustive task in the tradition which may require many changes to network configurations. The main issue is the proper translation of those high-level policies toward the low-level rules that satisfy all these SLA requirements (to satisfy business level goals). Therefore, the SDN controller should identify all requirements and resources needed to satisfy these SLA refinements, and monitor if these enforced low-level rules meet the requirement of the high-level policy all in an automatic way. The authors in [291] proposed a three stages procedure in SDN for SLA refinement. The first stage is manually done by the administrator to identify parameters or objectives (SLOs) From the SLA. Each service is pinned to a QoS class (e.g., QoS class such as platinum treatment of VoIP flows) that are stored in an LDAP repository. This repository contains requirements for each QoS class (e.g., delay, bandwidth) and their corresponding values (e.g., 200ms,128kbps). Also, it contains a list of commonly known protocols and there known ports that are stored by the administrator. In the second stage, the administrator associates protocol those stored in the LDAP repository with each service to differentiate its flows (e.g., VoIP service that its protocol will receive platinum QoS). In the third stage, the SDN controller loads these rules stored in the LDAP repository associated with those QoS class and protocols and store them into policy dictionary. For each network flow, the controller performs an analysis to implement low-level rules and enforce them on the network elements to apply QoS requirements for that applications or services (e.g., VoIP service). Authors depend mainly on manual work done by the administrator to do the translation from SLA to application requirements that later enforces as low-level rules in network elements that satisfy those requirements. This procedure needs to be automated to support autonomic QoS on SDN.

In autonomic QoS, users' QoS requirements are translated automatically into low-level network policy if they have the right of that and the network can handle it (e.g., the user pays extra money for extra Video quality or it been agreed before in SLA document). Users usually do not know about networks parameters. Therefore authors in [292] implemented a northbound API in the SDN controller to allow developers of video streaming application to enforce QoS requirements such as bandwidth, delay and so on. They are motivated by the importance of reducing the decoupling of rule-based code injection to do network configuration from the application code. They compared there work with existing modules for configuring or pushing QoS in Floodlight SDN controller that is configured manually. They argue that better bandwidth achieved when using their API by video streaming application develops instead of existing command-based tools for QoS configuration.

Similarly, the authors in [293], [294] proposed an SDN-based architecture to support the need or Real-Time Online Interactive Applications (ROIA) users such as gaming applications. Such applications depend on the input from users to determine the next state of the application or the service and then reply to users with the changes or updates of the application states. Therefore, the satisfaction of users in these services depends on the responsiveness in which loss and delay of action (i.e., transferred through packets) is a significant concern. Reservation of resources to guarantee the quality of service is not practical due to the static nature or inability to scale to large of users flows as it is the case in RSVP. Therefore, these user or application' needs should be specified dynamically to the network. Authors used the SDN due to its dynamic configuration through northbound API to accommodate these needs. Authors implemented an SDN module (within their proposed architrave) that is responsible for the communication between the SDN controller and the ROIA process that resides in the application server. Users are connected to this process and keep communicating with it during the session time. The application requirement such as response time is transmitted as QoS policy for either one or aggregated flows between the ROIA process and ROIA client(s) toward the SDN controller through the north API. The SDN controller uses statistics about these flows. The required QoS by the application is translated into QoS network metrics which the application or the user does not understand such as the jitter or throughput and so on. Even this work does not specify agreed SLA or user interactions, however, they implement an architecture in which an application specifies its requirements to the SDN network which is translated into a change of the network configuration which was inapplicable in a traditional network.

Users are not aware of these network QoS parameters; however, their satisfaction is a significant issue of business success. Quality of Experience is terminology that discusses the QoS from the user' satisfaction point of view. Which means guarantees of the QoS parameters may not satisfy the user. Therefore, this what is called user-centric QoS rather than network-centric which a significant challenge to service providers. The other challenge is how to map QoE into network-centric QoS parameters and to in how to dynamically specify these configurations based on QoE needs. The authors in [295] target these challenges in the field of IP Multimedia Subsystem (IMP) while designing an architecture for IPTV service. They use users rating of the services to reconfigure QoS network parameters. A QoE engine at the client side tries to learn about end user by using different profiles (action movie, talk show) within the ongoing time session and the user rating of the quality of the service. A prediction of the user's level of satisfaction is mapped into QoS network parameters base on these collected data (users rating) and the category of the service. They used linear regression to map between QoE into QoS parameters which been reflected into changes on the network rules that are enforced through the OpenFlow protocol. In that paper, the authors tried to convert the user's satisfaction level or what is called QoE into QoS parameters to improve the service. There is no translation or monitoring of SLA parameters within this work. However, it works for service providers that their goal is user happiness.

Similarly, the authors in [296] tried to give a score for QoE in which lower score means terrible services at the user side. The used an agent that monitors the service on the user side (e.g., IPTV networks or Video services such as Netflix) by recording network parameters such bandwidth, jitter and delay from the received packets on the user side. Then it pushes these data to the service provider side where the QoE score is computed where actions are taken on the SDN controller to improve the QoS parameters for that user(s) flows. It is more related to the user's satisfaction without direct interaction between the user, but its application is aware of the network. The authors provide only design without testing their proposed architecture. In [193], the authors propose a video streaming based on SDN (VSDN). The architecture allows the QoS application to request resources from the controller. It is a reservation based proposal by which a sender can request QoS for a specific video specification through QoS API to allow the controller to reserve the resources.

VI. DISCUSSION AND FUTURE DIRECTIONS

In this work, we review and survey exiting literature in the field of autonomic QoS support. We discuss and study work related to the MAPE-K model which is a well-known IBM model to describe Autonomic Systems. In the paper, we adopt MAPE-K mode as a reference model for its simplicity, clarity, modularity in describing how QoS autonomic provisioning and management should work. Key notes are given wherever appropriate. In this section, we summarize and discuss open research problems for the scientific community to address in order to improve the autonomic QoS management in SDN.

A. IMPROVING/OPTIMIZING QOS/QOE MONITORING

In measurements collection, there is always a trade-off between accuracy and collection costs (i.e., communication or TCAM memory). Such cost increases with the network size and traffic volume. Therefore, the optimization of the measurement process is an essential issue that needs more investigation. For instance, efficient adaptive measurement or sampling could help since timing is a significant factor in the accuracy and cost game. Adaption according to network state and emerging events and learning from network behavior history would help in producing optimized polling schemes. Moreover, an optimized polling scheme could consider the information source dimension besides timing. An optimized switch selection schemes should cover all traffic flows and reduces statistics messages exchanged between the control and data plane taking into account switches limited resources.

Another essential aspect that may affect measurements accuracy is traffic related to control and management, i.e. non-payload or non-data traffic. Network management packets for link discovery, DHCP and many others that get processed by switches. Such management packers are counted in some statistics, especially link related ones. The authors in [96], [97] pointed out that problem and tried to estimate it and remove that noise from data traffic measurements. Moreover, control messages need to be monitored besides the data traffic, either to obtain the volume of that traffic or to improve the network visibility to operators. Monitoring control and non-data traffic is still an open issue where it merely discussed by researchers for the monitoring module in SDN.

Introduction of models that can support the measurement process in which with the low sampling rate, the big picture of network state can be drawn still an open problem. For instance, authors in [175], derive a queue delay model from network parameters such as queue buffer size, queue bandwidth, number of flows, link propagation delay. Then, an estimated average queue delay is obtained from that model and used to control the end-to-end flows delay, and similarly, authors in [297] did. The most notable part is that such models can use maintained and measured network parameters by the controller and monitoring model can depend on. Especially in cases where statistics collection is costly, or network changes are infrequent. Moreover, most of such models do not consider user-centric QoS parameters that should be reflected in network-centric QoS or vice versa.

Beside network state information, application-awareness and customer profiling is another aspect that can improve QoS/QoE management in SDN networks. It requires the network to get information such as resource requirements from service providers or customer sides. Moreover, inspecting information such as session start and end time or service requirements from service providers which requires standardization of such APIs. Moreover, QoE at the user side is still an open research issue.

B. MACHINE LEARNING INCORPORATION IN QoS MANAGEMENT

Self-adaption to network changes and the autonomic support of QoS in SDN require intelligent behavior from the network. A simple threshold-based analysis is not helpful to improve QoS performance. More attention should be applied to usage of online analysis and Machine Learning. Few of the reviewed research works employ some ML techniques for the prediction of future trends in network traffic or QoS/QoE parameters. Analysis and Plan functions resemble the brain and muscles of the autonomic QoS management and ML can add more intelligence in their functionality. For instance, patterns of service degradation can be detected using such techniques if well-trained models are used in conjunction with collected switches statistics [155].

ML has the advantage of fast decision making without the need for sophisticated network models since it can treat the whole network as a black box. For example, ML techniques can be used to build more intelligent QoS routing. Moreover, ML can help in optimizing resource utilization and customer resource assignment decisions. Learning from customers behavior can help the network adapt its resource according to customers and services requirements. Also, its learning ability from actions taken by the network operator to resolve issues help in exploiting human experience in the form of machine defined policies that can resolve the same or similar future problem autonomically. However, ML has training issues that require large datasets that span long time intervals and which techniques should be uses needs more investigation in the field of QoS support.

C. AUTONOMIC POLICY-BASED QoS MANAGEMENT

Policy-based management that involves SLA enforcement into network policies and smart violation detection/prediction engines that get benefit of existing machine learning techniques is a major issue towards improving the autonomic behavior of the network or more specifically the QoS management in our case. However, few works that implement such features. Moreover, policy-based control of other functions (i.e. *Monitor, Analysis, Plan, Execute*) is limited in the literature. Most of the reviewed works assume a specific scenario (e.g. Monitor) just to improve QoS/QoE performance for a certain application without considering the application of autonomy features or the automation of such management tasks. Furthermore, in most of the cases, policies are operator defined without participation of machine inferred policies that can be defined from mining users/network history information that may improve the system performance and refine those policies. Especially considering the time dimension to improve the daily actions and plans for network operation by exploiting expert knowledge to define action profiles or templates to automate the management tasks. Authors of *GolfEngine* [268] try to define and implement a well-structured system that uses policy-based management of the *Monitor*, *Analysis* functions along with policies that control actions taken by the system. The system also offers policy conflict detection which is a major advantage since consistency between the data and control planes is a major issue since many applications could program the network. Therefore, a central view and automated control of all policies need to be implemented. The literature lack existence of similar systems that autonomically manage QoS for SDN network which require more efforts.

D. AUTONOMIC MULTI-DOMAIN END-TO-END QoS PROVISIONING

Most of discussed assume single domain QoS provisioning which is not the case in real Internet. User packets travel from one domain to another which usually controlled by different entities (e.g., various and may competitive ISPs) – each with its own QoS, billing policies. Therefore, multi-domain routing is another issue, especially when considering end-to-end QoS provisioning. In [220], the authors generalized their local domain LARAC-based QoS routing to address the distributed multi-domain problem. The path between two domain border nodes is modeled as a virtual link. Therefore, the link cost and delay are computed according to a network of virtual paths and border nodes. However, this can get more complicated that this since packets are treated based on policies implement between neighbor domains or ASes. Usually, a long-term SLAs are agreed among those entities to forward traffic along borders, and a change to such SLA requires more time to negotiate and comply, and implementation of such changes. With the introduction of SDN with its agility, flexibility, and programmability and the benefit of autonomic management of each domain to achieve the end-to-end QoS guarantees in conjunction with SDIXPs [62], [63] in between ASes [60], controlled by the SDN controller that can automatically reflect policies changes instead of the manual configuration used with BGP [59] that can cause errors [64] or as authors in [61] did when they implemented BGP as an SDN application, called *SDN-IP*. However, this requires great deal of information exchange between those evolved entities, either service provider, ISPs which makes it more challenging process since each entity may become more conservative toward sharing their internal configuration or state information directly or through global brokers as the case in *FlowBorker* [298]. Network resources can be shared since there is an expected benefit on return. Authors in [278] provide a cascade billing model to share the revenue of QoS guaranteed between the infrastructure provider and the service or content providers. These providers will be more incentivized to share their networks, especially between those ASes that do not collaborate or there is no agreement between them to carry data. This will also pose another challenge in monitoring and enforcing end-to-end QoS guarantees among all involved entities.

E. DISTRIBUTED/MULTIPLE CONTROL FOR QoS GUARANTEES

Few works discuss multiple controller support for QoS guarantees, and when they did, they use it for other purposes such as multi-domain routing as in [220], [221], or coordination network monitoring as in [90], [91]. OpenFlow 1.4 specification adds a new feature in which a controller can monitor changes on a subset of the flow table entries in case multiple controllers manage the network. Redundancy of control can help in reducing the load on the controller and for availability purposes. Fast, scalable QoS requests processing without service interruption is the aim of autonomic QoS provisioning. However, it brings the policy or SLA implementation consistency challenge in both control and data plane between different network users either for QoS support or any other applied network policies. Moreover, a shared knowledge source in the autonomic system should be consistent between those controllers images, and it may be affected by the flat/hierarchal or the slave-master relation between those controllers and the amount of information exchanged in between. Therefore, it is still an open issue and requires further investigation.

VII. CONCLUSION

In this article, we have reviewed the current state of literature on SDN to support end-to-end QoS guarantees, considering an SDN-managed network as an autonomic system that can provide and facilitate QoS functionalities. We classified the reviewed work using autonomic MAPE-K reference model. We discussed each function as stand alone, and focused on existing work relevant to satisfying the autonomy requirement. From our review and analysis in this article, there are many research areas in which autonomic-based QoS provisioning can be further enhanced. For example, SDN-based network monitoring was studied in the literature with no measurement for path or link delays. Also, little or no work was performed with respect to QoS routing. For example, LARAC algorithm is known to be a very efficient algorithm and is widely used for QoS routing. It is also possible to leverage ML techniques to build more intelligent QoS routing, as well as much more powerful analysis functions for autonomic QoS provisioning. The literature still lacks serious work on applying the autonomy features such as the self-management or policy-based QoS management with no human involvement.

ACKNOWLEDGMENT

The authors would like to acknowledge the support of King Fahd University of Petroleum and Minerals for this research.

REFERENCES

[1] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff, "A brief history of the Internet," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 5, pp. 22–31, 2009.

[2] *Cisco Visual Networking Index: Forecast and Methodology, 2016–2021*, Cisco, San Jose, CA, USA, 2017.

[3] R. Balakrishnan, *Advanced QoS for Multi-Service IP/MPLS Networks*. Hoboken, NJ, USA: Wiley, 2008.

[4] R. Braden, D. Clark, and S. Shenker. (1994). *Integrated Services in the Internet Architecture: An Overview*. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1633.txt

[5] S. Blake. (1998). *An Architecture for Differentiated Services*. [Online]. Available: https://tools.ietf.org/html/rfc2475

[6] R. Braden, D. Clark, and S. Shenker. (1997). *Resource ReSerVation Protocol (RSVP)*. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2205.txt

[7] *IETF Group*. Accessed: Jan. 5, 2019. [Online]. Available: https://www.ietf.org/

[8] ONF. *Open Networking Foundation*. Accessed: Jan. 5, 2019. [Online]. Available: https://www.opennetworking.org/

[9] ONF. (2013). *SDN Architecture Overview*. Accessed: Jan. 5, 2019. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf

[10] N. Feamster and J. Rexford, "Why (and how) networks should run themselves," 2017, *arXiv:1710.11583*. [Online]. Available: https://arxiv.org/abs/1710.11583

[11] M. Howard. (2016). *RESEARCH NOTE—75 Percent of Carriers Surveyed Have Deployed or Will Deploy SDN This Year—IHS Technology*. [Online]. Available: https://technology.ihs.com/583348

[12] M. Howard. (2016). *RESEARCH NOTE—75 Percent of Carriers Surveyed Have Deployed or Will Deploy SDN This Year—IHS Technology*. Accessed: May 27, 2018, [Online]. Available: https://technology.ihs.com/583348

[13] 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.

[14] N. McKeown, 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.

[15] Z. Zhao, E. Schiller, E. Kalogeiton, T. Braun, B. Stiller, M. T. Garip, J. Joy, M. Gerla, N. Akhtar, and I. Matta, "Autonomic communications in software-driven networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2431–2445, Nov. 2017.

[16] P.-W. Tsai, C.-W. Tsai, C.-W. Hsu, and C.-S. Yang, "Network monitoring in software-defined networking: A review," *IEEE Syst. J.*, vol. 12, no. 4, pp. 3958–3969, Dec. 2018.

[17] A. Yassine, H. Rahimi, and S. Shirmohammadi, "Software defined network traffic measurement: Current trends and challenges," *IEEE Instrum. Meas. Mag.*, vol. 18, no. 2, pp. 42–50, Apr. 2015. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7066685

[18] D. Kreutz, F. M. V. 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.

[19] I. M. Alsmadi, I. Alazzam, and M. Akour, "A systematic literature review on software-defined networking," in *Information Fusion for Cyber-Security Analytics*, vol. 691. Cham, Switzerland: Springer, 2017, pp. 333–369.

[20] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed SDN controller system: A survey on design choice," *Comput. Netw.*, vol. 121, pp. 100–111, Jul. 2017.

[21] A. Mirchev, "Survey of Concepts for QoS improvements via SDN," in *Proc. Future Internet Innov. Internet Technol. Mobile Commun. (IITM)*, vol. 33, 2015, p. 1.

[22] M. Karakus and A. Durresi, "Quality of service (QoS) in software defined networking (SDN): A survey," *J. Netw. Comput. Appl.*, vol. 80, pp. 200–218, Feb. 2016.

[23] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[24] Z. Movahedi, M. Ayari, R. Langar, and G. Pujolle, "A survey of autonomic network architectures and evaluation criteria," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 2, pp. 464–490, 2nd Quart., 2012.

[25] J. O. Kephart and W. E. Walsh, "An artificial intelligence perspective on autonomic computing policies," in *Proc. 5th IEEE Int. Workshop Policies Distrib. Syst. Netw. (POLICY)*, Jun. 2004, pp. 3–12.

[26] P. Horn. (2001). *Autonomic Computing: IBM's Perspective on the State of Information Technology*. IBM J. Pap. [Online]. Available: http://people.scs.carleton.ca/~soma/biosec/readings/autonomic_computing.pdf

[27] J. R. McBride, A. R. Lupini, M. A. Schreuder, N. J. Smith, S. J. Pennycook, and S. J. Rosenthal, "An architectural blueprint for autonomic computing," IBM White Paper, Jun. 2006, pp. 1–6.

[28] M. Parashar and S. Hariri, *Autonomic Computing: An Overview*. Cham, Switzerland: Springer, 2005, pp. 257–269.

[29] R. Sterritt, "Autonomic computing," *Innov. Syst. Softw. Eng.*, vol. 1, no. 1, pp. 79–88, 2005.

[30] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing—Degrees, models, and applications," *ACM Comput. Surv.*, vol. 40, no. 3, 2008, Art. no. 7.

[31] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing—Degrees, models, and applications," *ACM Comput. Surv.*, vol. 40, no. 3, 2008, Art. no. 7.

[32] B. Khemka, R. Friese, L. D. Briceño, H. J. Siegel, A. A. Maciejewski, G. A. Koenig, C. Groer, G. Okonski, M. M. Hilton, R. Rambharos, and S. Poole, "Utility functions and resource management in an oversubscribed heterogeneous computing environment," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2394–2407, Aug. 2015.

[33] D. M. Chess, A. Segal, I. Whalley, and S. R. White, "Unity: Experiences with a prototype autonomic computing system," in *Proc. Int. Conf. Auton. Comput.*, May 2004, pp. 140–147.

[34] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White, "A multi-agent systems approach to autonomic computing," in *Proc. 3rd Int. Joint Conf. Auton. Agents Multiagent Syst.*, vol. 1, Jul. 2004, pp. 464–471.

[35] J. O. Kephart and R. Das, "Achieving self-management via utility functions," *IEEE Internet Comput.*, vol. 11, no. 1, pp. 40–48, Jan./Feb. 2007.

[36] J. Case, M. Fedor, M. Schoffstall, and J. Davin, *Simple Network Management Protocol (SNMP)*, document RFC 1157, 1990.

[37] J. Moy, *OSPF Version 2. Internet Request For Comments*, document RFC 2328, Apr. 1998. [Online]. Available: https://tools.ietf.org/html/rfc2328

[38] C. Jelger, C. Tschudin, S. Schmid, and G. Leduc, "Basic abstractions for an autonomic network architecture," in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimedia Netw. (WOWMOM)*, Jun. 2007, pp. 1–6.

[39] G. Bouabene, C. Jelger, C. Tschudin, S. Schmid, A. Keller, and M. May, "The autonomic network architecture (ANA)," *IEEE J. Sel. Areas Commun.*, vol. 28, no. 1, pp. 4–14, Jan. 2010.

[40] H. Derbel, N. Agoulmine, and M. Salaün, "ANEMA: Autonomic network management architecture to support self-configuration and self-optimization in IP networks," *Comput. Netw.*, vol. 53, no. 3, pp. 418–430, 2009.

[41] *Open vSwitch*. Accessed: Jan. 5, 2019. [Online]. Available: http://www.openvswitch.org/

[42] *OpenFlow 1.3 Software Switch*. Accessed: Jan. 5, 2019. [Online]. Available: https://github.com/CPqD/ofsoftswitch13

[43] Indigo. *Project Floodlight*. Accessed: Jan. 5, 2019. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/Indigo/overview

[44] *Arista 7150*. Accessed: Jan. 5, 2019. [Online]. Available: https://www.arista.com/assets/data/pdf/Datasheets/7150S_Datasheet.pdf

[45] *NEC PF5240 Switch*. Accessed: Jan. 5, 2019. [Online]. Available: https://www.necam.com/sdn/Hardware/PF5240Switch/

[46] *Pica8 P-3297*. Accessed: Jan. 5, 2019. [Online]. Available: http://www.pica8.org/products/pre-loaded-switches

[47] 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.

[48] *POX*. Accessed: Jan. 5, 2019. [Online]. Available: https://github.com/noxrepo/pox

[49] Z. Cai, "Maestro: Achieving scalability and coordination in centralizaed network control plane," Ph.D. dissertation, Rice Univ., Houston, TX, USA, 2012.

[50] D. Erickson, "The beacon openflow controller," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, Aug. 2013, pp. 13–18.

[51] *The Beacon OpenFlow Controller*. Accessed: Jan. 5, 2019. [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home

[52] *Floodlight*. Accessed: Jan. 5, 2019. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[53] *RYU*. Accessed: Jan. 5, 2019. [Online]. Available: http://osrg.github.io/ryu/

[54] *RYU SDN Framework, Release 1.0*, NTT Commun., South Korea, 2014.

[55] *OpenDaylight*. Accessed: Jan. 5, 2019. [Online]. Available: https://www.opendaylight.org/

[56] *ONOS*. Accessed: Jan. 5, 2019. [Online]. Available: https://onosproject.org/

[57] *IRIS (OpenIRIS)*. Accessed: Jan. 5, 2019. [Online]. Available: http://openiris.etri.re.kr/

[58] B. Lee, S. H. Park, J. Shin, and S. Yang, "IRIS: The Openflow-based Recursive SDN controller," in *Proc. 16th Int. Conf. Adv. Commun. Technol. (ICACT)*, Feb. 2014, pp. 1227–1231.

[59] Y. Rekhter, T. Li, and S. Hares, *A Border Gateway Protocol 4 (BGP-4)*, document RFC 4271, 2005.

[60] M. E. Tozal, "The Internet: A system of interconnected autonomous systems," in *Proc. Annu. IEEE Syst. Conf. (SysCon)*, Orlando, FL, USA, Apr. 2016, pp. 1–8. doi: 10.1109/SYSCON.2016.7490628.

[61] *SDN-IP*. Accessed: Jan. 5, 2019. [Online]. Available: https://wiki.onosproject.org/display/ONOS/SDN-IP

[62] 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," in *Proc. ACM Conf. SIGCOMM (SIGCOMM)*, 2014, pp. 551–562.

[63] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever, "An industrial-scale software defined Internet exchange point," in *Proc. NSDI*, 2016, pp. 1–14.

[64] N. Feamster and H. Balakrishnan, "Detecting BGP configuration faults with static analysis," in *Proc. 2nd Conf. Symp. Netw. Syst. Design Implement.*, vol. 2, 2005, pp. 43–56.

[65] B. Edgeworth, D. Prall, J. M. Barozet, A. Lockhart, and N. Ben-Dvora, *Cisco Intelligent WAN (IWAN)*. San Jose, CA, USA: Cisco Press, 2016.

[66] *Cisco Application Policy Infrastructure Controller Enterprise Module*. Accessed: Jan. 5, 2019. [Online]. Available: https://www.cisco.com/c/en/us/products/cloud-systems-management/application-policy-infrastructure-controller-enterprise-module/index.html

[67] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An intellectual history of programmable networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, 2014.

[68] B. Renukadevi and S. D. M. Raja, "Deep packet inspection management application in SDN," in *Proc. 2nd Int. Conf. Comput. Commun. Technol. (ICCCT)*, Feb. 2017, pp. 256–259.

[69] S. Seeber, L. Stiemert, and G. D. Rodosek, "Towards an SDN-enabled IDS environment," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Florence, Italy, Sep. 2015, pp. 751–752.

[70] M. A. Sayeed, M. A. Sayeed, and S. Saxena, "Intrusion detection system based on software defined network firewall," in *Proc. 1st Int. Conf. Next Gener. Comput. Technol. (NGCT)*, Dehradun, India, Sep. 2015, pp. 379–382.

[71] Y. Hande, A. Muddana, and S. Darade, "Software-defined network-based intrusion detection system," in *Innovations in Electronics and Communication Engineering* (Lecture Notes in Networks and Systems), vol. 7. Singapore: Springer, 2018, pp. 535–543.

[72] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 393–430, 1st Quart., 2019.

[73] V. Heorhiadi, M. K. Reiter, and V. Sekar, "Simplifying software-defined network optimization using SOL," in *Proc. NSDI*, 2018, pp. 223–237.

[74] S. Khan, A. Gani, A. A. Wahab, M. Guizani, and M. K. Khan, "Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 303–324, 1st Quart., 2017.

[75] *OF-CONFIG 1.2, OpenFlow Management and Configuration Protocol*, Open Network Found., Menlo Park, CA, USA, 2014.

[76] *OpenFlow Switch Specification 1.0*, Open Network Foundation, Dec. 2009.

[77] *OpenFlow Switch Specification 1.1*, Open Network Foundation, Menlo Park, CA, USA, Feb. 2011.

[78] *OpenFlow Switch Specification 1.2*, Open Network Foundation, Menlo Park, CA, USA, Dec. 2011.

[79] *OpenFlow Switch Specification 1.3*, Open Network Foundation, Menlo Park, CA, USA, Jun. 2012.

[80] *OpenFlow Switch Specification 1.4*, Open Network Foundation, Menlo Park, CA, USA, Oct. 2013.

[81] *OpenFlow Switch Specification 1.5*, Open Network Foundation, Menlo Park, CA, USA, Dec. 2014.

[82] M. Hartung and M. Körner, "SOFTmon—Traffic monitoring for SDN," *Procedia Comput. Sci.*, vol. 110, pp. 516–523, Jun. 2017.

[83] B. Siniarski, J. Murphy, and D. Delaney, "FlowVista: Low-bandwidth SDN monitoring driven by business application interaction," in *Proc. 25th Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM)*, Sep. 2017, pp. 1–6.

[84] A. Gangwal, M. Conti, and M. S. Gaur, "Panorama: Real-time bird's eye view of an openflow network," in *Proc. IEEE 14th Int. Conf. Netw., Sens. Control (ICNSC)*, May 2017, pp. 204–209.

[85] M. Aslan and A. Matrawy, "Maintaining an up-to-date global network view in SDN," 2016, *arXiv:1612.04944*. [Online]. Available: https://arxiv.org/abs/1612.04944

[86] R. B. Santos, T. R. Ribeiro, and C. D. A. César, "A network monitor and controller using only openflow," in *Proc. 8th Latin Amer. Netw. Oper. Manage. Symp. (LANOMS)*, Oct. 2015, pp. 9–16. [Online]. Available http://ieeexplore.ieee.org/abstract/document/7332663/

[87] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic matrix estimator for openflow networks," in *Passive and Active Measurement*, vol. 6032. Berlin, Germany: Springer, 2010, p. 201.

[88] M. Singh, N. Varyani, J. Singh, and K. Haribabu, *Estimation of End-to-End Available Bandwidth and Link Capacity in SDN*. Cham, Switzerland: Springer, 2017. [Online]. Available: https://books.google.com

[89] S. Rowshanrad, S. Namvarasl, and M. Keshtgari, "A queue monitoring system in openflow software defined networks," *J. Telecommun. Inf. Technol.*, vol. 2017, no. 1, pp. 39–43, 2017.

[90] H. Tahaei, R. Salleh, S. Khan, R. Izard, K. K. R. Choo, and N. B. Anuar, "A multi-objective software defined network traffic measurement," *Measurement*, vol. 95, pp. 317–327, Jan. 2017. doi: 10.1016/j.measurement.2016.10.026.

[91] H. Tahaei, R. B. Salleh, M. F. A. Razak, K. Ko, and N. B. Anuar, "Cost effective network flow measurement for software defined networks: A distributed controller scenario," *IEEE Access*, vol. 6, pp. 5182–5198, 2018. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/8245797/

[92] C. Liu, A. Malboubi, and C.-N. Chuah, "OpenMeasure: Adaptive flow measurement & inference with online learning in SDN," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 47–52.

[93] M. Moshref, M. Yu, and R. Govindan, "Resource/accuracy tradeoffs in software-defined measurement," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 73–78.

[94] L. Jose, M. Yu, and J. Rexford, "Online measurement of large traffic aggregates on commodity switches," in *Proc. USENIX HotICE Work.*, 2011, p. 13.

[95] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "FlowCover: Low-cost flow monitoring scheme in software defined networks," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, Dec. 2014, pp. 1956–1961. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7037094/. doi: 10.1109/GLOCOM.2014.7037094.

[96] Y. Sinha, S. Vashishth, and K. Haribabu, "Real time monitoring of packet loss in software defined networks," in *Ubiquitous Communications and Network Computing* (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), vol. 218. Cham, Switzerland: Springer, 2018, pp. 154–164.

[97] Y. Sinha, S. Vashishth, and K. Haribabu, *Estimation of Raw Packets in SDN*. Cham, Switzerland: Springer, 2017. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-73423-1_13

[98] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "CeMon: A cost-effective flow monitoring system in software defined networks," *Comput. Netw.*, vol. 92, pp. 101–115, Dec. 2015. doi: 10.1016/j.comnet.2015.09.018.

[99] J. A. P. Fernández, L. J. G. Villalba, and T.-H. Kim, "Clustering and flow conservation monitoring tool for software defined networks," *Sensors*, vol. 18, no. 4, p. 1079, 2018. [Online]. Available: http://www.mdpi.com/1424-8220/18/4/1079

[100] J. A. P. Fernández and L. J. G. Villalba. *FloCon: Flow Conservation Framework for Monitoring SDN Networks*. Accessed: Jan. 5, 2019. [Online]. Available: https://www.waset.org/downloads/17/papers/17fr050539.pdf

[101] N. Grover, N. Agarwal, and K. Kataoka, "LiteFlow: Lightweight and distributed flow monitoring platform for SDN," in *Proc. 1st IEEE Conf. Netw. Softwarization, Softw.-Defined Infrastruct. Netw., Clouds, IoT Services (NETSOFT)*, Apr. 2015, pp. 1–9.

[102] Z. Yang and K. L. Yeung, "An efficient flow monitoring scheme for SDN networks," in *Proc. IEEE 30th Can. Conf. Elect. Comput. Eng.*, Apr./May 2017, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7946633/

[103] J. Tang, Y. Zhang, and Y. Li, "AdaRate: A rate-adaptive traffic measurement method in software defined networks," *Int. J. Performability Eng.*, vol. 13, no. 6, pp. 937–944, 2017. [Online]. Available: http://ijpe-online.com/adarate-a-rate-adaptive-traffic-measurement-method-in-software-defined-networks.html

[104] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in openflow software-defined networks," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–8.

[105] D.-E. Henni, Y. Hadjaj-Aoul, and A. Ghomari, "Probe-SDN: A smart monitoring framework for SDN-based networks," in *Proc. Global Inf. Infrastruct. Netw. Symp.*, Oct. 2016, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7814940/

[106] Q. He and S. Wang, "A low-cost measurement framework in software defined networks," *Int. J. Commun. Netw. Syst. Sci.*, vol. 10, no. 5, pp. 54–66, 2017. [Online]. Available: http://file.scirp.org/pdf/IJCNS_2017052709354034.pdf and http://www.scirp.org/journal/doi.aspx. doi: 10.4236/ijcns.2017.105B006.

[107] Q. He, X. Wang, and M. Huang, "OpenFlow-based low-overhead and high-accuracy SDN measurement framework," *Trans. Emerg. Telecommun. Technol.*, vol. 29, no. 2, 2018, Art. no. e3263.

[108] C. Yu, C. Lumezanu, A. Sharma, Q. Xu, G. Jiang, and H. V. Madhyastha, "Software-defined latency monitoring in data center networks," in *Passive and Active Measurement* (Lecture Notes in Computer Science), vol. 8995. Cham, Switzerland: Springer, 2015, pp. 360–372.

[109] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for software defined networks," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–9.

[110] G. Cheng and J. Yu, "Adaptive sampling for openflow network measurement methods," in *Proc. 12th Int. Conf. Future Internet Technol.*, 2017, Art. no. 4.

[111] S. Shirali-Shahreza and Y. Ganjali, "FleXam: Flexible sampling extension for monitoring and security applications in openflow," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 167–168.

[112] V. Altukhov and E. Chemeritskiy, "On real-time delay monitoring in software-defined networks," in *Proc. Int. Sci. Technol. Conf. (Mod. Netw. Technol.) (MoNeTeC)*, Oct. 2014, pp. 1–6.

[113] M. Selmchenko, M. Beshley, O. Panchenko, and M. Klymash, "Development of monitoring system for end-to-end packet delay measurement in software-defined networks," in *Proc. 13th Int. Conf. Mod. Problems Radio Eng., Telecommun. Comput. Sci.*, Feb. 2016, pp. 667–670.

[114] S. Ramanathan, Y. Kanza, and B. Krishnamurthy, "SDProber: A software defined prober for SDN," in *Proc. Symp. SDN Res.(SOSR)*, Mar. 2018, Art. no. 17. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3185467.3185472

[115] W. Zhang, X. Zhang, H. Shi, and L. Zhou, "An efficient latency monitoring scheme in software defined networks," *Future Gener. Comput. Syst.*, vol. 83, pp. 303–309, Jun. 2018. doi: 10.1016/j.future.2018.01.033.

[116] A. M. Allakany and K. Okamura, "Latency monitoring in software-defined networks," in *Proc. 12th Int. Conf. Future Internet Technol.*, 2017, Art. no. 5. [Online]. Available: https://dl.acm.org/citation.cfm?id=3095791

[117] L. Nacshon, R. Puzis, and P. Zilberman, "Floware: Balanced flow monitoring in software defined networks," 2016, *arXiv:1608.03307*. [Online]. Available: https://arxiv.org/abs/1608.03307

[118] R. Puzis, L. Nacshon, and P. Zilberman, "NetFlow for openflow (NFO): Balanced flow monitoring in software defined networks," *IEEE Trans. J. Name*, no. 1, pp. 1–14, 2016.

[119] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "FlowSense: Monitoring network utilization with zero measurement cost," in *Passive and Active Measurement* (Lecture Notes in Computer Science), vol. 7799. Berlin, Germany: Springer, 2013, pp. 31–41.

[120] B. Wang, J. Su, J. Li, and B. Han, "EffiView: Trigger-based monitoring approach with low cost in SDN," in *Proc. IEEE 19th Int. Conf. High Perform. Comput. Commun. (HPCC)*, Dec. 2017, pp. 309–315.

[121] *sFlow*. Accessed: Jan. 5, 2019. [Online]. Available: https://sflow.org/

[122] *NetFlow*. Accessed: Jan. 5, 2019. [Online]. Available: https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html

[123] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "OpenSample: A low-latency, sampling-based measurement platform for commodity SDN," in *Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst.*, Jun./Jul. 2014, pp. 228–237.

[124] J. Suárez-Varela and P. Barlet-Ros, "Reinventing netflow for openflow software-defined networks," 2017, *arXiv:1702.06803*. [Online]. Available: https://arxiv.org/abs/1702.06803

[125] J. Suárez-Varela and P. Barlet-Ros, "Towards a netflow implementation for openflow software-defined networks," in *Proc. 29th Int. Teletraffic Congr. (ITC)*, vol. 1, Sep. 2017, pp. 187–195.

[126] J. Suárez-Varela and P. Barlet-Ros, "SBAR: SDN flow-Based monitoring and Application Recognition," in *Proc. Symp. SDN Res.*, 2018, Art. no. 22.

[127] M.-H. Chen, Y.-C. Tien, Y.-T. Huang, I.-H. Chung, and C.-F. Chou, "A low-latency two-tier measurement and control platform for commodity SDN," *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 98–104, Sep. 2016.

[128] P. H. A. Rezende, P. R. S. L. Coelho, L. F. Faina, L. J. Camargos, and R. Pasquini, "Analysis of monitoring and multipath support on top of openflow specification," *Int. J. Netw. Manag.*, vol. 28, no. 3, May 2018, Art. no. e2017.

[129] Y. Afek, A. Bremler-Barr, S. L. Feibish, and L. Schiff, "Sampling and large flow detection in SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 345–346, 2015. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2829988.2790009

[130] Y. Afek, A. Bremler-Barr, S. L. Feibish, and L. Schiff, "Detecting heavy flows in the SDN match and action model," *Comput. Netw.*, vol. 136, pp. 1–12, May 2018.

[131] W. Kim, J. Li, J. W.-K. Hong, and Y.-J. Suh, "OFMon: OpenFlow monitoring system in ONOS controllers," in *Proc. IEEE NetSoft Conf. Workshops*, Jun. 2016, pp. 397–402. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7502474/

[132] A. Adegboyega, "An adaptive resource provisioning scheme for effective QoS maintenance in the IaaS cloud," in *Proc. Int. Work. Virtualization Technol.*, 2011, pp. 2:1–2:6. doi: 10.1145/2835075.2835078.

[133] A. Rego, A. Canovas, J. M. Jiménez, and J. Lloret, "An intelligent system for video surveillance in IoT environments," *IEEE Access*, vol. 6, pp. 31580–31598, 2018.

[134] T. Hafeez, N. Ahmed, B. Ahmed, and A. W. Malik, "Detection and mitigation of congestion in SDN enabled data center networks: A survey," *IEEE Access*, vol. 6, pp. 1730–1740, 2017.

[135] M. Afaq, S. U. Rehman, and W.-C. Song, "Visualization of elephant flows and QoS provisioning in SDN-based networks," in *Proc. 17th Asia–Pacific Netw. Oper. Manag. Symp. Manag. (APNOMS)*, Aug. 2015, pp. 444–447. doi: 10.1109/APNOMS.2015.7275384.

[136] J. Liu, J. Li, G. Shou, Y. Hu, Z. Guo, and W. Dai, "SDN based load balancing mechanism for elephant flow in data center networks," in *Proc. Int. Symp. Wireless Pers. Multimedia Commun. (WPMC)*, Sep. 2015, pp. 486–490. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7014867/

[137] Y.-H. Huang, W.-Y. Shih, and J.-L. Huang, "A classification-based elephant flow detection method using application round on SDN environments," in *Proc. 19th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Sep. 2017, pp. 231–234. [Online]. Available: http://ieeexplore.ieee.org/document/8094140/

[138] S. Bashir and N. Ahmed, "VirtMonE: Efficient detection of elephant flows in virtualized data centers," in *Proc. 25th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Nov. 2015, pp. 280–285. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7366826/

[139] R. B. Basat, G. Einziger, R. Friedman, and Y. Kassner, "Optimal elephant flow detection," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.

[140] Z. Liu, D. Gao, Y. Liu, and H. Zhang, "An enhanced scheduling mechanism for elephant flows in SDN-based data center," in *Proc. IEEE 84th Veh. Technol. Conf.*, Sep. 2016, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7880900/

[141] C. Xing, K. Ding, C. Hu, and M. Chen, "Sample and fetch-based large flow detection mechanism in software defined networks," *IEEE Commun. Lett.*, vol. 20, no. 9, pp. 1764–1767, Sep. 2016.

[142] P. Xiao, W. Qu, H. Qi, Y. Xu, and Z. Li, "An efficient elephant flow detection with cost-sensitive in SDN," in *Proc. 1st Int. Conf. Ind. Netw. Intell. Syst.*, Mar. 2015, pp. 24–28.

[143] L. Yang, B. Ng, and W. K. G. Seah, "Heavy hitter detection and identification in software defined networking," in *Proc. 25th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2016, pp. 1–10. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7568527/

[144] C. Bi, X. Luo, T. Ye, and Y. Jin, "On precision and scalability of elephant flow detection in data center with SDN," in *Proc. IEEE Globecom Work. GC Wkshps*, Dec. 2013, pp. 1227–1232. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6825161/

[145] Z. Wang, C. Zhou, Y. Yu, X. Shi, X. Yin, and J. Yao, *Fast Detection of Heavy Hitters in Software Defined Networking Using an Adaptive and Learning Method* (Lecture Notes in Computer Science), vol. 11065. Cham, Switzerland: Springer, 2018. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-00012-7_5

[146] B. Wang and J. Su, "A survey of elephant flow detection in SDN," in *Proc. 6th Int. Symp. Digit. Forensic Secur. (ISDFS)*, Antalya, Turkey, Mar. 2018, pp. 1–6.

[147] A. Al-Jawad, P. Shah, O. Gemikonakli, and R. Trestian, "LearnQoS: A learning approach for optimizing QoS over multimedia-based SDNs," in *Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcast. (BMSB)*, Jun. 2018, pp. 1–6.

[148] A. Sabbeh, Y. Al-Dunainawi, H. S. Al-Raweshidy, and M. F. Abbod, "Performance prediction of software defined network using an artificial neural network," in *Proc. SAI Comput. Conf. (SAI)*, Jul. 2016, pp. 80–84.

[149] S. Shakeri, S. Parsaeefard, and M. Derakhshani, "Proactive admission control and dynamic resource management in SDN-based virtualized networks," in *Proc. 8th Int. Conf. Netw. Future (NOF)*, Nov. 2018, pp. 46–51.

[150] C. Chen-Xiao and X. Ya-Bin, "Research on load balance method in SDN," *Int. J. Grid Distrib. Comput.*, vol. 9, no. 1, pp. 25–36, 2016.

[151] A. B. Letaifa, "Adaptive QoE monitoring architecture in SDN networks: Video streaming services case," in *Proc. 13th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2017, pp. 1383–1388.

[152] A. B. Letaifa, "Real time ML-based QoE adaptive approach in SDN context for HTTP video services," *Wireless Pers. Commun.*, vol. 103, no. 3, pp. 2633–2656, Dec. 2018.

[153] T. Abar, A. B. Letaifa, and S. El Asmi, "Enhancing QoE based on machine learning and DASH in SDN networks," in *Proc. 32nd IEEE Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, May 2018, pp. 258–263.

[154] T. Abar, A. B. Letaifa, and S. El Asmi, "Machine learning based QoE prediction in SDN networks," in *Proc. 13th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2017, pp. 1395–1400.

[155] R. Stadler, R. Pasquini, and V. Fodor, "Learning from network device statistics," *J. Netw. Syst. Manage.*, vol. 25, no. 4, pp. 672–698, Oct. 2017.

[156] J. Bendriss, I. G. B. Yahia, and D. Zeghlache, "Forecasting and anticipating SLO breaches in programmable networks," in *Proc. 20th Conf. Innov. Clouds, Internet Networks (ICIN)*, Mar. 2017, pp. 127–134.

[157] J. Bendriss, I. G. B. Yahia, P. Chemouil, and D. Zeghlache, "AI for SLA management in programmable networks," in *Proc. 13th Int. Conf. Design Reliable Commun. Netw.*, 2017, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7993445/

[158] R. Pasquini and R. Stadler, "Learning end-to-end application QoS from openflow switch statistics," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Jul. 2017, pp. 1–9.

[159] A. Mestres, E. Alarcón, Y. Ji, and A. Cabellos-Aparicio, "Understanding the modeling of computer network delays using neural networks," in *Proc. Workshop Big Data Anal. Mach. Learn. Data Commun. Netw. (Big-DAMA)*, 2018, pp. 46–52.

[160] A. Azzouni, R. Boutaba, and G. Pujolle, "NeuRoute: Predictive dynamic routing for software-defined networks," Sep. 2017, *arXiv:1709.06002*. [Online]. Available: https://arxiv.org/abs/1709.06002

[161] A. Azzouni and G. Pujolle, "NeuTM: A neural network-based framework for traffic matrix prediction in SDN," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2018, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/8406199/

[162] S. Jain, M. Khandelwal, A. Katkar, and J. Nygate, "Applying big data technologies to manage QoS in an SDN," in *Proc. 12th Int. Conf. Netw. Service Manage. (CNSM)*, 2017, pp. 302–306.

[163] T. Abar, A. B. Letaifa, and S. El Asmi, "How modeling QoE requirements using game theory," in *Proc. 32nd IEEE Int. Conf. Adv. Inf. Netw. Workshops (WAINA)*, May 2018, pp. 147–152.

[164] A. B. Letaifa, G. Maher, and S. Mouna, "ML based QoE enhancement in SDN context: Video streaming case," in *Proc. 13th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, 2017, pp. 103–108.

[165] C. Zhao, H. Chang, and Q. Liu, "Bayesian algorithm based traffic prediction of big data services in OpenFlow controlled optical networks," in *Proc. IEEE 2nd Int. Conf. Big Data Anal. (ICBDA)*, Mar. 2017, pp. 823–826. [Online]. Available: http://ieeexplore.ieee.org/document/8078752/

[166] M. Alreshoodi and J. Woods, "An empirical study based on a fuzzy logic system to assess the QoS/QoE correlation for layered video streaming," in *Proc. IEEE Int. Conf. Comput. Intell. Virtual Environ. Meas. Syst. Appl. (CIVEMSA)*, Jul. 2013, pp. 180–184. [Online]. Available: http://ieeexplore.ieee.org/document/6617417/

[167] M. Alreshoodi, J. Woods, and I. K. Musa, "Optimising the delivery of scalable H.264 video stream by QoS/QoE correlation," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2015, pp. 253–254. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7066401/

[168] J. Park, S. M. Raza, P. Thorat, D. S. Kim, and H. Choo, "Network traffic prediction model based on training data," in *Computational Science and Its Applications* (Lecture Notes in Computer Science), vol. 9158. Cham, Switzerland: Springer, 2015, pp. 117–127. [Online]. Available: http://link.springer.com/10.1007/978-3-319-21410-8_9

[169] J. Pang, G. Xu, X. Fu, and K. Zhao, "Horizon: A QoS management framework for SDN-based data center networks," *Ann. Telecommun.*, vol. 72, nos. 9–10, pp. 597–605, 2017. [Online]. Available: https://link.springer.com/article/10.1007/s12243-017-0579-2

[170] Y. Han, T. Jeong, J.-H. Yoo, and J. W.-K. Hong, "FLAME: Flow level traffic matrix estimation using poisson shot-noise process for SDN," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Jun. 2016, pp. 102–106. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7502453/

[171] H. Z. Jahromi, A. Hines, and D. T. Delanev, "Towards application-aware networking: ML-based end-to-end application KPI/QoE metrics characterization in SDN," in *Proc. 10th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, 2018, pp. 126–131. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8436625/

[172] P. H. Isolani, J. A. Wickboldt, C. B. Both, J. Rochol, and L. Z. Granville, "Interactive monitoring, visualization, and configuration of OpenFlow-based SDN," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2015, pp. 207–215.

[173] D. Pajin and P. V. Vuletić, "OF2NF: Flow monitoring in OpenFlow environment using NetFlow/IPFIX," in *Proc. 1st IEEE Conf. Netw. Softwarization (NetSoft)*, Apr. 2015, pp. 1–5.

[174] D. Sinha, K. Haribabu, and S. Balasubramaniam, "Real-time monitoring of network latency in software defined networks," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst. (ANTS)*, Dec. 2015, pp. 1–3.

[175] H. Ma, J. Yan, P. Georgopoulos, and B. Plattner, "Towards SDN based queuing delay estimation," *China Commun.*, vol. 13, no. 3, pp. 27–36, 2016.

[176] M. Malboubi, L. Wang, C.-N. Chuah, and P. Sharma, "Intelligent SDN based traffic (de) aggregation and measurement paradigm (iSTAMP)," in *Proc. IEEE INFOCOM*, Apr./May 2014, pp. 934–942.

[177] X. T. Phan and K. Fukuda, "SDN-Mon: Fine-grained traffic monitoring framework in software-defined networks," *J. Inf. Process.*, vol. 25, pp. 182–190, Feb. 2017.

[178] X. T. Phan and K. Fukuda, "SDN-Mon: Fine-grained traffic monitoring framework in software-defined networks," *J. Inf. Process.*, vol. 25, pp. 182–190, 2017.

[179] X. T. Phan and K. Fukuda, "Toward a flexible and scalable monitoring framework in software-defined networks," in *Proc. 31st IEEE Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, Mar. 2017, pp. 403–408.

[180] S. Kim, G. Yang, C. Yoo, and S. Min, "BFD-based link latency measurement in software defined networking," in *Proc. 13th Int. Conf. Netw. Service Manage. (CNSM)*, 2017, pp. 2–7.

[181] X. Zhang, Y. Wang, J. Zhang, L. Wang, and Y. Zhao, "A two-way link loss measurement approach for software-defined networks," in *Proc. IEEE/ACM 25th Int. Symp. Qual. Service (IWQoS)*, Jun. 2017, pp. 1–10. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7969164/

[182] B. Siniarski, C. Olariu, P. Perry, and J. Murphy, "OpenFlow based VoIP QoE monitoring in enterprise SDN," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 660–663.

[183] K. Phemius and M. Bouet, "Monitoring latency with OpenFlow," in *Proc. 9th Int. Conf. Netw. Service Manage. (CNSM)*, 2013, pp. 122–125.

[184] C. Thorpe, A. Hava, J. Langlois, A. Dumas, and C. Olariu, "iMOS: Enabling VoIP QoS monitoring at intermediate nodes in an OpenFlow SDN," in *Proc. IEEE Int. Conf. Cloud Eng. Workshop (IC2EW)*, Berlin, Germany, Apr. 2016, pp. 76–8. doi: 10.1109/IC2EW.2016.33.

[185] D.-H. Luong, A. Outtagarts, and A. Hebbar, "Traffic monitoring in software defined networks using opendaylight controller," in *Mobile, Secure, and Programmable Networking* (Lecture Notes in Computer Science), vol. 10026, 2016, pp. 38–48.

[186] X. Zhang, W. Hou, L. Guo, S. Wang, Q. Zhang, P. Guo, and R. Li, "Joint optimization of latency monitoring and traffic scheduling in software defined heterogeneous networks," *Quality, Reliability, Security and Robustness in Heterogeneous Systems* (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), vol. 234. 2018. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-319-78078-8.pdf#page=113

[187] Y. Li, Z.-P. Cai, and H. Xu, "LLMP: Exploiting LLDP for latency measurement in software-defined data center networks," *J. Comput. Sci. Technol.*, vol. 33, no. 2, pp. 277–285, Mar. 2018.

[188] S. Skiena, *Implementing Discrete Mathematics: Combinatorics and Graph Theory With Mathematica*. Reading, MA, USA: Addison-Wesley, 1991, pp. 225–227.

[189] S. Ren, Q. Feng, and W. Dou, "An end-to-end QoS routing on software defined network based on hierarchical token bucket queuing discipline," in *Proc. Int. Conf. Data Mining, Commun. Inf. Technol. (DMCIT)*, 2017, Art. no. 3.

[190] A. Kucminski, A. Al-Jawad, P. Shah, and R. Trestian, "QoS-based routing over software defined networks," in *Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcast. (BMSB)*, Jun. 2017, pp. 1–6.

[191] M. Karl, J. Gruen, and T. Herfet, "Multimedia optimized routing in OpenFlow networks," in *Proc. IEEE Int. Conf. Netw. (ICON)*, Dec. 2013, pp. 1–6.

[192] H. Owens, A. Durresi, and R. Jain, "Reliable video over software-defined networking (RVSDN)," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2014, pp. 1974–1979.

[193] H. Owens, II, and A. Durresi, "Video over software-defined networking (VSDN)," *Comput. Netw.*, vol. 92, pp. 341–356, Dec. 2015.

[194] Y. Yue, Y. Ran, S. Chen, B. Yang, L. Sun, and J. Yang, "Joint routing and layer selecting for scalable video transmission in SDN," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2015, pp. 1–6.

[195] C. Yu, C. Lumezanu, A. Sharma, Q. Xu, G. Jiang, and H. V. Madhyastha, "Software-defined latency monitoring in data center networks," in *Proc. Int. Conf. Passive Act. Netw. Meas.*, 2015, pp. 360–372. [Online]. Available: http://link.springer.com/10.1007/978-3-319-15509-8_27

[196] T. M. Thi, T. Huynh, and W.-J. Hwang, "QoS-enabled streaming of multiple description coded video over OpenFlow-based networks," *Nonlinear Theory Appl. (IEICE)*, vol. 6, no. 2, pp. 144–159, 2015.

[197] J. Yan, H. Zhang, Q. Shuai, B. Liu, and X. Guo, "HiQoS: An SDN-based multipath QoS solution," *China Commun.*, vol. 12, no. 5, pp. 123–133, May 2015.

[198] S. T. V. Pasca, S. S. P. Kodali, and K. Kataoka, "AMPS: Application aware multipath flow routing using machine learning in SDN," in *Proc. 23rd Nat. Conf. Commun. (NCC)*, 2017, pp. 1–6.

[199] D. Eppstein, "Finding the k shortest paths," *SIAM J. Comput.*, vol. 28, no. 2, pp. 652–673, 1998.

[200] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "QoS-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun./Jul. 2016, pp. 25–33.

[201] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntes-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," Sep. 2017. *arXiv:1709.07080*. [Online]. Available: https://arxiv.org/abs/1709.07080

[202] C. Yu, J. Lan, Z. Guo, and Y. Hu, "DROM: Optimizing the routing in software-defined networks with deep reinforcement learning," *IEEE Access*, vol. 6, pp. 64533–64539, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8502806/

[203] Y.-S. Yu and C.-H. Ke, "Genetic algorithm-based routing method for enhanced video delivery over software defined networks," *Int. J. Commun. Syst.*, vol. 31, no. 1, p. e3391, 2018. [Online]. Available: http://doi.wiley.com/10.1002/dac.3391

[204] D. Cavendish and G. Mario, "Internet QoS routing using the Bellman-Ford algorithm," in *High Performance Networking*. Springer, Boston, MA, USA, 1998, pp. 627–646.

[205] N.-F. Huang, I.-J. Liao, H.-W. Liu, S.-J. Wu, and C.-S. Chou, "A dynamic QoS management system with flow classification platform for software-defined networks," in *Proc. 8th Int. Conf. Ubi-Media Comput. (UMEDIA)*, 2015, pp. 72–77.

[206] Á. L. V. Caraguay, J. A. P. Fernández, and L. J. G. Villalba, "Framework for optimized multimedia routing over software defined networks," *Comput. Netw.*, vol. 92, pp. 369–379, Dec. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128615003230

[207] A. M. Al-Sadi, A. Al-Sherbaz, J. Xue, and S. Turner, "Routing algorithm optimization for software defined network WAN," in *Proc. Al-Sadeq Int. Conf. Multidisciplinary IT Commun. Sci. Appl.*, 2016, pp. 1–6.

[208] S. Shamim and Z. Fei, "Evaluating a QoS aware path selection service using the GENI network," in *Proc. IEEE Int. Conf. Ubiquitous Wireless Broadband (ICUWB)*, Oct. 2016, pp. 1–4.

[209] S. Tomovic and I. Radusinovic, "Fast and efficient bandwidth-delay constrained routing algorithm for SDN networks," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Jun. 2016, pp. 303–311.

[210] A. Gangwal, M. Gupta, M. S. Gaur, V. Laxmi, and M. Conti, "ELBA: Efficient layer based routing algorithm in SDN," in *Proc. Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2016, pp. 1–7.

[211] M.-T. Kao, B. Huang, S. Kao, and H. Tseng, "An effective routing mechanism for link congestion avoidance in software-defined networking," in *Proc. Int. Comput. Symp. (ICS)*, 2016, pp. 154–158.

[212] D.-E. Henni, A. Ghomari, and Y. Hadjadj-Aoul, "Videoconferencing over OpenFlow networks: An optimization framework for QoS routing," in *Proc. IEEE Int. Conf. Comput. Inf. Technol.; Ubiquitous Comput. Commun.; Dependable, Autonomic Secure Comput.; Pervasive Intell. Comput.*, Oct. 2015, pp. 491–496.

[213] S. Civanlar, M. Parlakisik, A. M. Tekalp, B. Gorkemli, B. Kaytaz, and E. Onem, "A QoS-enabled OpenFlow environment for scalable video streaming," in *Proc. IEEE Globecom Workshops (GC)*, Dec. 2010, pp. 351–356.

[214] H. E. Egilmez, S. Civanlar, and A. M. Tekalp, "An optimization framework for QoS-enabled adaptive video streaming over OpenFlow networks," *IEEE Trans. Multimedia*, vol. 15, no. 3, pp. 710–715, Apr. 2013.

[215] H. E. Egilmez, B. Gorkemli, A. M. Tekalp, and S. Civanlar, "Scalable video streaming over OpenFlow networks: An optimization framework for QoS routing," in *Proc. Int. Conf. Image Process. (ICIP)*, 2011, pp. 2241–2244.

[216] C. Xu, B. Chen, P. Fu, and H. Qian, "A dynamic resource allocation model for guaranteeing quality of service in software defined networking based cloud computing environment," in *Cloud Computing and Security* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, 2015, pp. 206–217.

[217] A. Juttner, B. Szviatovski, I. Mecs, and Z. Rajko, "Lagrange relaxation based method for the QoS routing problem," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2001, pp. 859–868.

[218] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *Proc. Asia–Pacific Signal Inf. Process. Assoc. Annu. Summit Conf. (APSIPA ASC)*, 2012, pp. 1–8.

[219] S. Tomovic, N. Prasad, and I. Radusinovic, "SDN control framework for QoS provisioning," in *Proc. 22nd Telecommun. Forum Telfor (TELFOR)*, 2014, pp. 111–114.

[220] H. E. Egilmez, S. Civanlar, and A. M. Tekalp, "A distributed QoS routing architecture for scalable video streaming over multi-domain OpenFlow networks," in *Proc. 19th IEEE Int. Conf. Image Process.*, Sep./Oct. 2012, pp. 2237–2240. [Online]. Available: http://ieeexplore.ieee.org/document/6467340/

[221] H. E. Egilmez and A. M. Tekalp, "Distributed QoS architectures for multimedia streaming over software defined networks," *IEEE Trans. Multimedia*, vol. 16, no. 6, pp. 1597–1609, Oct. 2014.

[222] W.-E. Liang and C.-A. Shen, "A high performance media server and QoS routing for SVC streaming based on software-defined networking," in *Proc. Int. Conf. Comput. Netw. Commun. (ICNC)*, 2017, pp. 556–560.

[223] T.-F. Yu, K. Wang, and Y.-H. Hsu, "Adaptive routing for video streaming with QoS support over SDN networks," in *Proc. Int. Conf. Inf. Netw.*, 2015, pp. 318–323.

[224] J. W. Guck, M. Reisslein, and W. Kellerer, "Model-based control plane for fast routing in industrial QoS network," in *Proc. IEEE 23rd Int. Symp. Qual. Service (IWQoS)*, Jun. 2016, pp. 65–66.

[225] J. W. Guck, M. Reisslein, and W. Kellerer, "Function split between delay-constrained routing and resource allocation for centrally managed QoS in industrial networks," *IEEE Trans. Ind. Informat.*, vol. 12, no. 6, pp. 2050–2061, Dec. 2016.

[226] G. Liu and K. G. Ramakrishnan, "A*Prune: An algorithm for finding K shortest paths subject to multiple constraints," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM), 20th Annu. Joint Conf. IEEE Comput. Commun. Soc.*, vol. 2, Apr. 2001, pp. 743–749.

[227] J. W. Guck and W. Kellerer, "Achieving end-to-end real-time quality of service with software defined networking," in *Proc. IEEE 3rd Int. Conf. Cloud Netw.*, Oct. 2014, pp. 70–76. [Online]. Available: http://ieeexplore.ieee.org/document/6968971/

[228] O. Dobrijevic, A. J. Kassler, L. Skorin-Kapov, and M. Matijašević, "Q-POINT: QoE-driven path optimization model for multimedia services," in *Proc. Int. Conf. Wired/Wireless Internet Commun.*, 2014, pp. 134–147.

[229] C. Lin, K. Wang, and G. Deng, "A QoS-aware routing in SDN hybrid networks," *Procedia Comput. Sci.*, vol. 110, pp. 242–249, Jan. 2017. doi: 10.1016/j.procs.2017.06.091.

[230] R. Kumar, M. Hasan, S. Padhy, K. Evchenko, L. Piramanayagam, S. Mohan, and R. B. Bobba, "End-to-end network delay guarantees for real-time systems using SDN," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 2018, pp. 231–242.

[231] O. Dobrijevic, M. Santl, and M. Matijasevic, "Ant colony optimization for QoE-centric flow routing in software-defined networks," in *Proc. 11th Int. Conf. Netw. Service Manag.*, Nov. 2015, pp. 274–278. [Online]. Available: http://ieeexplore.ieee.org/document/7367371/

[232] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, "Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 388–415, 1st Quart., 2018. doi: 10.1109/COMST.2017.2749760.

[233] S. Layeghy, F. Pakzad, and M. Portmann, "SCOR: Software-defined constrained optimal routing platform for SDN," Jul. 2016, *arXiv:1607.03243*. [Online]. Available: https://arxiv.org/abs/1607.03243

[234] S. Layeghy, F. Pakzad, and M. Portmann, "SCOR: Constraint programming-based northbound interface for SDN," in *Proc. 26th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, 2016, pp. 83–88.

[235] S. Layeghy, F. Pakzad, and M. Portmann, "A new QoS routing northbound interface for SDN," *Austral. J. Telecommun. Digit. Econ.*, vol. 4, no. 3, Sep. 2016.

[236] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "SDN-based application-aware networking on the example of Youtube video streaming," in *Proc. 2nd Eur. Workshop Softw. Defined Netw. (EWSDN)*, 2013, pp. 87–92.

[237] T. Zinner, M. Jarschel, A. Blenk, F. Wamser, and W. Kellerer, "Dynamic application-aware resource management using software-defined networking: Implementation prospects and challenges," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–6.

[238] A. V. Akella and K. Xiong, "Quality of service (QoS)-guaranteed network resource allocation via software defined networking (SDN)," in *Proc. IEEE 12th Int. Conf. Dependable, Autonomic Secure Comput.*, Aug. 2014, pp. 7–13.

[239] C. Xu, B. Chen, and H. Qian, "Quality of service guaranteed resource management dynamically in software defined network," *J. Commun.*, vol. 10, no. 11, pp. 843–850, 2015.

[240] H. Cui, C. Ma, W. Lai, L. Zheng, and Y. Liu, "Accurate network resource allocation in SDN according to traffic demand," in *Proc. 4th Int. Conf. Mechatron., Mater., Chem. Comput. Eng. (ICMMCCE)*, 2015, pp. 1166–1175.

[241] R. M. Abuteir, A. Fladenmuller, and O. Fourmaux, "SDN based architecture to improve video streaming in home networks," in *Proc. Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Mar. 2016, pp. 220–226.

[242] T.-N. Lin, Y.-M. Hsu, S.-Y. Kao, and P.-W. Chi, "OpenE2EQoS: Meter-based method for end-to-end QoS of multimedia services over SDN," in *Proc. IEEE Int. Symp. Pers., Indoor, Mobile Radio Commun. (PIMRC)*, Sep. 2016, pp. 1–6.

[243] W. Miao, G. Min, Y. Wu, and H. Wang, "Performance modelling of preemption-based packet scheduling for data plane in software defined networks," in *Proc. IEEE Int. Conf. Smart City/SocialCom/SustainCom (SmartCity)*, Dec. 2015, pp. 60–65. [Online]. Available: http://ieeexplore.ieee.org/document/7463702/

[244] G. S. Aujla, R. Chaudhary, N. Kumar, R. Kumar, and J. J. P. C. Rodrigues, "An ensembled scheme for QoS-aware traffic flow management in software defined networks," in *Proc. IEEE Int. Conf. Commun.*, May 2018, pp. 1–7.

[245] H.-C. Chu and T.-H. Lin, "An adaptive user-defined traffic control mechanism for SDN," in *Mobile and Wireless Technologies* (Lecture Notes in Electrical Engineering), vol. 425. Singapore: Springer, 2018, pp. 609–619.

[246] T.-W. Chu, C.-A. Shen, and C.-W. Wu, "The hardware and software co-design of a configurable QoS for video streaming based on OpenFlow protocol and NetFPGA platform," *Multimedia Tools Appl.*, vol. 77, no. 7, pp. 9071–9091, 2018.

[247] C. Caba and J. Soler, "SDN-based QoS aware network service provisioning," *Mobile, Secure, and Programmable Networking* (Lecture Notes in Computer Science), vol. 9395. Cham, Switzerland: Springer, 2015, pp. 119–133.

[248] M.-T. Thi, T. Huynh, M. Hasegawa, and W.-J. Hwang, "A rate allocation framework for multi-class services in software-defined networks," *J. Netw. Syst. Manage.*, vol. 25, no. 1, pp. 1–20, 2017.

[249] M. M. Tajiki, B. Akbari, and N. Mokari, "QRTP: QoS-Aware resource reallocation based on traffic prediction in software defined cloud networks," in *Proc. 8th Int. Symp. Telecommun. (IST)*, 2016, pp. 527–532.

[250] M. M. Tajiki, B. Akbari, M. Shojafar, and N. Mokari, "Joint QoS and congestion control based on traffic prediction in SDN," *Appl. Sci.*, vol. 7, no. 12, p. 1265, 2017. [Online]. Available: http://www.mdpi.com/2076-3417/7/12/1265

[251] T. Feng, J. Bi, and K. Wang, "Allocation and scheduling of network resource for multiple control applications in SDN," *China Commun.*, vol. 12, no. 6, pp. 85–95, 2015.

[252] F. Tao, J. Bi, and K. Wang, "Allocation and scheduling of network resource for multiple control applications in SDN," *China Commun.*, vol. 12, no. 6, pp. 85–95, 2015.

[253] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium*, vol. 1, p. 132, Oct. 2009.

[254] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "OpenVirteX: Make your virtual SDNs programmable," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 25–30.

[255] L. Liao, V. C. M. Leung, and P. Nasiopoulos, "DFVisor: Scalable network virtualization for QoS management in cloud computing," in *Proc. 10th Int. Conf. Netw. Service Manag. (CNSM)*, 2015, pp. 328–331.

[256] L. Liao, A. Shami, and V. C. M. Leung, "Distributed FlowVisor: A distributed FlowVisor platform for quality of service aware cloud network virtualisation," *IET Netw.*, vol. 4, pp. 270–277, Sep. 2015.

[257] Z. Bozakov and P. Papadimitriou, "AutoSlice: Automated and scalable slicing for software-defined networks," in *Proc. ACM Conf. CoNEXT Student Workshop*, 2012, pp. 3–4.

[258] W. Khumngoen, W. Putthividhya, and V. Tan-Anannuwat, "Fine-grained bandwidth allocation in software-defined networks," in *Proc. 20th Int. Comput. Sci. Eng. Conf. (ICSEC)*, 2017, pp. 1–6.

[259] C.-W. Tseng, Y.-K. Huang, Y.-T. Yang, C.-C. Liu, D.-Y. Huang, and L.-D. Chou, "A network traffic shunt system in SDN network," in *Proc. Int. Conf. Comput., Inf. Telecommun. Syst. (CITS)*, Jul. 2017, pp. 195–199. [Online]. Available: http://ieeexplore.ieee.org/document/8035281/

[260] H. Thanh, A. V. Vu, D. L. Nguyen, M. N. Tran, Q. T. Ngo, T.-H. Truong, T. H. Nguyen, and T. Magedanz, "A generalized resource allocation framework in support of multi-layer virtual network embedding based on SDN," *Comput. Netw.*, vol. 92, pp. 251–269, Dec. 2015.

[261] A. X. Porxas, S.-C. Lin, and M. Luo, "QoS-aware virtualization-enabled routing in software-defined networks," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2015, pp. 5771–5776.

[262] B. Pfaff and B. Davie, *The Open VSwitch Database Management Protocol*, document RFC 7047, 2013. doi: 10.17487/rfc7047.

[263] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, and P. Yalagandula, "Automated and scalable QoS control for network convergence," in *Proc. Internet Netw. Manag. Conf. Res. Enterprise Netw.*, 2010, p. 1.

[264] M. Capelle, S. Abdellatif, M.-J. Huguet, and P. Berthou, "Online virtual links resource allocation in software-defined networks," in *Proc. 14th IFIP Netw. Conf.*, 2015, pp. 1–9.

[265] R. Trivisonno, R. Guerzoni, I. Vaishnavi, and A. Frimpong, "Network resource management and QoS in SDN-enabled 5G systems," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–7.
[266] S. D'Oro, L. Galluccio, P. Mertikopoulos, G. Morabito, and S. Palazzo, "Auction-based resource allocation in OpenFlow multi-tenant networks," *Comput. Netw.*, vol. 115, pp. 29–41, Mar. 2017.
[267] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about SDN flow tables," in *Proc. Int. Conf. Passive Act. Netw. Meas.* Cham, Switzerland: Springer, 2015, pp. 347–359.
[268] Q. Li, R. Mohammadi, M. Conti, C. Li, and X. Li, "GolfEngine: Network management system for software defined networking," in *Proc. 13th IEEE Int. Conf. Intell. Comput. Commun. Process. (ICCP)*, Sep. 2017, pp. 239–246.
[269] Y. Liu, Y. Li, Y. Wang, and J. Yuan, "Optimal scheduling for multi-flow update in software-defined networks," *J. Netw. Comput. Appl.*, vol. 54, pp. 11–19, Aug. 2015.
[270] P. Perešíni, M. Kuzniar, M. Canini, and D. Kostic, "ESPRES: Easy scheduling and prioritization for SDN," *Open Netw. Summit Res. Track*, pp. 2–3, Mar. 2014.
[271] P. Peresíni, M. Ku, M. Canini, and D. Kostić "ESPRES: Transparent SDN update scheduling," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 73–78.
[272] L. Wang, Q. Li, Y. Jiang, Y. Wang, R. Sinnott, and J. Wu, "IRMS: An intelligent rule management scheme for software defined networking," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Atlanta, GA, USA, May 2017, pp. 742–747.
[273] B. Halder, M. S. Barik, and C. Mazumdar, "Detection of flow violation in distributed SDN controller," in *Proc. 5th Int. Conf. Emerg. Appl. Inf. Technol. (EAIT)*, 2018, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8470417/
[274] K.-T. Foerster, S. Schmid, and S. Vissicchio, "Survey of consistent software-defined network updates," 2018, *arXiv:1609.02305*. [Online]. Available: https://arxiv.org/abs/1609.02305
[275] R. Wallner and R. Cannistra, "An SDN approach: Quality of service using big switch's floodlight open-source controller," in *Proc. Asia–Pacific Adv. Netw.*, vol. 35, 2013, p. 14.
[276] D. Palma, J. Goncalves, B. Sousa, L. Cordeiro, P. Simoes, S. Sharma, and D. Staessens, "The QueuePusher: Enabling queue management in OpenFlow," in *Proc. 3rd Eur. Workshop Softw.-Defined Netw. (EWSDN)*, 2014, pp. 125–126.
[277] S. Dwarakanathan, L. Bass, and L. Zhu, "Cloud application HA using SDN to ensure QoS," in *Proc. IEEE 8th Int. Conf. Cloud Comput. (CLOUD)*, Jun./Jul. 2015, pp. 1003–1007.
[278] S. Sharma, D. Palma, J. Goncalves, D. Staessens, N. Johnson, C. Palansuriya, R. Figueiredo, L. Cordeiro, D. Morris, A. Carter, R. Baxter, D. Colle, and M. Pickavet, "CityFlow, enabling quality of service in the Internet: Opportunities, challenges, and experimentation," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 272–280.
[279] C. Caba and J. Soler, "APIs for QoS configuration in software defined networks," in *Proc. 1st IEEE Conf. Netw. Softwarization (NetSoft)*, Apr. 2015, pp. 1–5.
[280] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelém, "Control of multiple packet schedulers for improving QoS on OpenFlow/SDN networking," in *Proc. 2nd Eur. Workshop Softw. Defined Netw. (EWSDN)*, 2013, pp. 81–86.
[281] M. S. Seddiki, M. Shahbaz, S. Donovan, S. Grover, M. Park, N. Feamster, and Y.-Q. S. Flowqos, "FlowQoS: Per-flow quality of service for broadband access networks," in *Proc. 3rd Work. Hot Top. Softw. Defin. Netw.-HotSDN*, 2014, pp. 207–208.
[282] I. N. Bozkurt, Y. Zhou, and T. Benson, "Dynamic prioritization of traffic in home networks," in *Proc. CoNEXT Student Workshop*, Berlin, Germany, Dec. 2015, pp. 1–3.
[283] G. Tangari, D. Tuncer, M. Charalambides, and G. Pavlou, "Decentralized Monitoring for Large-Scale Software-Defined Networks," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 289–297.
[284] W. Wassapon, P. Uthayopa, C. Chantrapornchai, and K. Ichikawa, "Real-time monitoring and visualization software for OpenFlow network," in *Proc. 15th Int. Conf. ICT Knowl. Eng.*, 2017, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8259622/
[285] *MongoDB: Open Source Document Database*. Accessed: Jan. 5, 2019. [Online]. Available: https://www.mongodb.com/
[286] W.-H. Hsu, X.-H. Wang, S.-C. Yeh, and P.-S. Huang, "The implementation of a QoS/QoE mapping and adjusting application in software-defined networks," in *Proc. 2nd Int. Conf. Intell. Green Building Smart Grid (IGBSG)*, 2016, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7539431/
[287] L. A. D. Knob, R. P. Esteves, L. Z. Granville, and L. M. R. Tarouco, "Mitigating elephant flows in SDN-based IXP networks," in *Proc. IEEE Symp. Comput. Commun.*, Jul. 2017, pp. 1352–1359. [Online]. Available: http://ieeexplore.ieee.org/document/8024712/
[288] A. Al-jawad, P. Shah, O. Gemikonakli, and R. Trestian, "Policy-based QoS management framework for software-defined networks," in *Proc. Int. Symp. Netw., Comput. Commun.*, 2016, pp. 1–6. [Online]. Available: http://eprints.mdx.ac.uk/id/eprint/24194
[289] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, P. Dely, "Towards QoE-driven multimedia service negotiation and path optimization with software defined networking," in *Proc. 20th Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM)*, 2012, pp. 1–5.
[290] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "PolicyCop: An autonomic QoS policy enforcement framework for software defined networks," in *Proc. Workshop Softw. Defined Netw. Future Netw. Services (SDN4FNS)*, Nov. 2013, pp. 1–7.
[291] C. C. Machado, L. Z. Granville, A. Schaeffer-Filho, and J. A. Wickboldt, "Towards SLA policy refinement for QoS management in software-defined networking," in *Proc. Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, May 2014, pp. 397–404.
[292] C. R. Vasconcelos, R. C. M. Gomes, A. F. B. F. Costa, and D. D. C. da Silva, "Enabling high-level network programming: A northbound API for Software-Defined Networks," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Da Nang, Vietnam, Jan. 2017, pp. 662–667.
[293] S. Gorlatch, T. Humernbrum, and F. Glinka, "Improving QoS in real-time Internet applications: From best-effort to software-defined networks," in *Proc. Int. Conf. Comput. Netw. Commun. (ICNC)*, Feb. 2014, pp. 189–193.
[294] S. Gorlatch and T. Humernbrum, "Enabling high-level QoS metrics for interactive online applications using SDN," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Feb. 2015, pp. 707–711.
[295] T. Huong-Truong, N. H. Thanh, N. T. Hung, J. Mueller, and T. Magedanz, "QoE-aware resource provisioning and adaptation in IMS-based IPTV using OpenFlow," in *Proc. 19th IEEE Workshop Local Metropolitan Area Netw. (LANMAN)*, Brussels, Belgium, Apr. 2013, pp. 1–3.
[296] H. Balwani, P. Wagh, R. Vadaje, S. Shivgunde, and P. M. S. Wakode, "Implementation of Qoe/QoS mapping inSDN," *Int. Res. J. Eng. Technol.*, vol. 4, no. 4, pp. 957–959, 2017.
[297] A. Iqbal, U. Javed, S. Saleh, J. Kim, J. S. Alowibdi, and M. U. Ilyas, "Analytical modeling of end-to-end delay in OpenFlow based networks," *IEEE Access*, vol. 5, pp. 6859–6871, 2017.
[298] D. Marconett and S. J. B. Yoo, "FlowBroker: Market-driven multi-domain SDN with heterogeneous brokers," in *Proc. Opt. Fiber Commun. Conf. Exhibit. (OFC)*, Los Angeles, CA, USA, 2015, pp. 1–3.

AHMED BINSAHAQ received the B.S. degree in computer and information technology from Yarmouk University, Irbid, Jordan, in 2010, and the M.S. degree in computer networks from the King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia, in 2015, where he is currently pursuing the Ph.D. degree in computer engineering. His research interests include WSNs, the IoT, resource allocation, and software-defined networks.

TAREK R. SHELTAMI received the Ph.D. degree in electrical and computer engineering from the Department of Electrical and Computer Engineering, Queen's University, Kingston, ON, Canada, in 2003. He was a Consultant in wireless networks with GamaEng Inc., Ottawa, ON, Canada, from 2002 to 2004. He also worked in several joint projects with Nortel Network Corporation, Ottawa. He is currently an Associate Professor with the Computer Engineering Department, King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia. He is also an Adjunct Professor with the Jodrey School of Computer Science, Acadia University, Wolfville, Canada. Before joining KFUPM, he was a Research Associate Professor with the School of Information Technology and Engineering, University of Ottawa, Ottawa. He has been a member of the Technical Program and Organizing Committees of several international IEEE conferences. He is the Co-Founder of the IEEE International Symposium on Emerging Ubiquitous and Pervasive Systems (EUPS) and the IEEE International Symposium on Applications of Ad hoc and Sensor Networks (AASNET).

KHALED SALAH received the B.S. degree in computer engineering with a minor in computer science from Iowa State University, USA, in 1990, and the M.S. degree in computer systems engineering and the Ph.D. degree in computer science from the Illinois Institute of Technology, USA, in 1994 and 2000, respectively. He joined Khalifa University, in 2010, and is teaching graduate and undergraduate courses in the areas of cloud computing, computer and network security, computer networks, operating systems, and performance modeling and analysis. Prior to joining Khalifa University, he worked for ten years at the Department of Information and Computer Science, King Fahd University of Petroleum and Minerals (KFUPM), Saudi Arabia. He is currently a Full Professor with the Department of Electrical and Computer Engineering, Khalifa University, United Arab Emirates. He serves on the editorial boards of many WOS-listed journals, including *IET Communications*, *IET Networks*, *Journal of Network and Computer Applications* (Elsevier), *Security and Communication Networks* (Wiley), *International Journal of Network Management* (Wiley), *Journal of Universal Computer Science*, and *Arabian Journal for Science and Engineering*.