SDN Controllers: A Comprehensive Analysis and Performance Evaluation Study

LIEHUANG ZHU, MD M. KARIM, KASHIF SHARIF, CHANG XU, and FAN LI,
Beijing Institute of Technology
XIAOJIANG DU, Temple University
MOHSEN GUIZANI, Qatar University

Software-defined networks offer flexible and intelligent network operations by splitting a traditional network into a centralized control plane and a programmable data plane. The controller in the control plane is the fundamental element used to manage all operations of the data plane. Hence, the performance and capabilities of the controller itself are essential in achieving optimal performance. Furthermore, the tools used to benchmark their performance must be accurate and useful in measuring different evaluation parameters. There are dozens of controller proposals for general and specialized networks in the literature. However, there is a very limited comprehensive quantitative analysis for them. In this article, we present a comprehensive qualitative comparison of different SDN controllers, along with a quantitative analysis of their performance in different network scenarios. We categorize and classify 34 controllers and present a qualitative comparison. We also present a comparative analysis of controllers for specialized networks such as the Internet of Things, blockchain networks, vehicular networks, and wireless sensor networks. We also discuss in-depth capabilities of benchmarking tools and provide a comparative analysis of their capabilities. This work uses three benchmarking tools to compare 9 controllers and presents a detailed analysis of their performance, along with discussion on performance of specialized network controllers.

CCS Concepts: • **General and reference → Cross-computing tools and techniques**; **Measurement**; **Metrics**; **Evaluation**; **Performance**; • **Networks → Network performance evaluation**;

Additional Key Words and Phrases: Software-defined networks, SDN controller, benchmarking tools, Internet of Things, blockchain, vehicular networks, wireless sensor networks

ACM Reference format:
Liehuang Zhu, Md M. Karim, Kashif Sharif, Chang Xu, Fan Li, Xiaojiang Du, and Mohsen Guizani. 2020. SDN Controllers: A Comprehensive Analysis and Performance Evaluation Study. *ACM Comput. Surv.* 53, 6, Article 133 (December 2020), 40 pages.
https://doi.org/10.1145/3421764

The work of L. Zhu was funded by the National Natural Science Foundation of China (grants 61872041 and U1836212).
Authors' addresses: L. Zhu, Md M. Karim, K. Sharif (corresponding author), C. Xu, and F. Li, Beijing Institute of Technology, 5 South Zhongguancun Street, Haidian District, Beijing, China, 100081; emails: {liehuangz, mkarim, kashif, xchang, fli}@bit.edu.cn; X. Du, Temple University, 1801 North Broad Street, Philadelphia, PA; email: dux@temple.edu; M. Guizani, Qatar University, P.O. Box 2713, Doha, Qatar; email: mguizani@ieee.org.

0360-0300/2020/12-ART133 $15.00
https://doi.org/10.1145/3421764

1 INTRODUCTION

Software-defined networks (SDNs) have seen tremendous growth and deployment in different types of networks in recent times. They are actively used in data center (DC) networks [82, 90], wireless and Internet of Things (IoT) networks [23, 79], and wide area and cellular networks [130], as well as security and privacy of domains [210]. Compared to traditional networks, they decouple the control logic from network layer devices and centralize it for efficient traffic forwarding and flow management across the domain. This multi-layered architecture, as shown in Figure 1(a), has data forwarding devices at the bottom in the data plane, which is programmed by controllers in the control plane. The high-level application or management plane interacts with a control layer to program the whole network and enforce different policies. The interaction among these layers functions as communication/programming protocols.

Traditional networks suffer from several limitations, mainly due to diverse service requirements and the scale of the network. Some of these are related to traffic engineering, flow management, policy enforcement, security, and virtualization. On the contrary, SDN presents a simplified, centralized, and efficient solution to these by decoupling the data plane forwarding and control plane intelligence. Hence, the network switches become simple forwarding devices, which route data traffic based on the instructions from a softwarized controller. This centralized entity provides programmatic control of the whole network and enables real-time control of underlying devices. Using SDN, network management becomes simpler and helps in removing network rigidity.

Some of the well-known controllers are NOX [77], POX [144], Floodlight [11], OpenDaylight (ODL) [113], Open Network Operating System (ONOS) [24], and Ryu [153]. However, many other controllers and flavors are available in the literature. From a practical implementation perspective, it is challenging to determine which controller will perform best in any given type of network. Hence, the qualitative and quantitative comparative analysis of these controllers is critical. Similarly, from a general control plane perspective, it is essential to evaluate whether the controller is capable of efficiently manage the complete network and utilize the capabilities of the data plane to its maximum capacity. Although the fundamental function of a controller is flow management and installation, several performance metrics can be used for its benchmarking. As there are numerous controllers available with different architectures and properties, it becomes imperative to have standard benchmarking criteria for evaluation. In this regard, there are two basic requirements: a set of benchmarking metrics and an efficient tool for benchmarking tests. Vengainathan et al. [185] have presented an elementary list of tests that should be conducted to evaluate the performance of a controller. However, there can be other metrics that should also be used when benchmarking different controllers. Similarly, the tool used to perform the test in an emulated environment is critical [106].

Scope of this study. The scope of this work is highlighted in Figure 2. SDNs have been studied from different aspects, but the two main types of studies can be classified as *general SDN surveys* such as [103, 117, 194], which broadly cover the domain and many of its different components, and *SDN controller studies*, which are more targeted to SDN controllers only. Within the controller-specific studies, some works have theoretically explained the different controllers and their properties from different aspects [18, 52, 136, 207]. However, the majority of these studies only describe the architecture and properties of a few controllers, yet the list of controllers presented in the literature is very long. The other studies that are controller specific evaluate the performance of controllers using testbeds for actual quantitative performance. We discuss them in detail in Section 5.

This work extends both subcategories, as shown in Figure 2, to a great extent. First, we analyze 34 controllers for different properties (Section 3), which to the best of our knowledge has not been

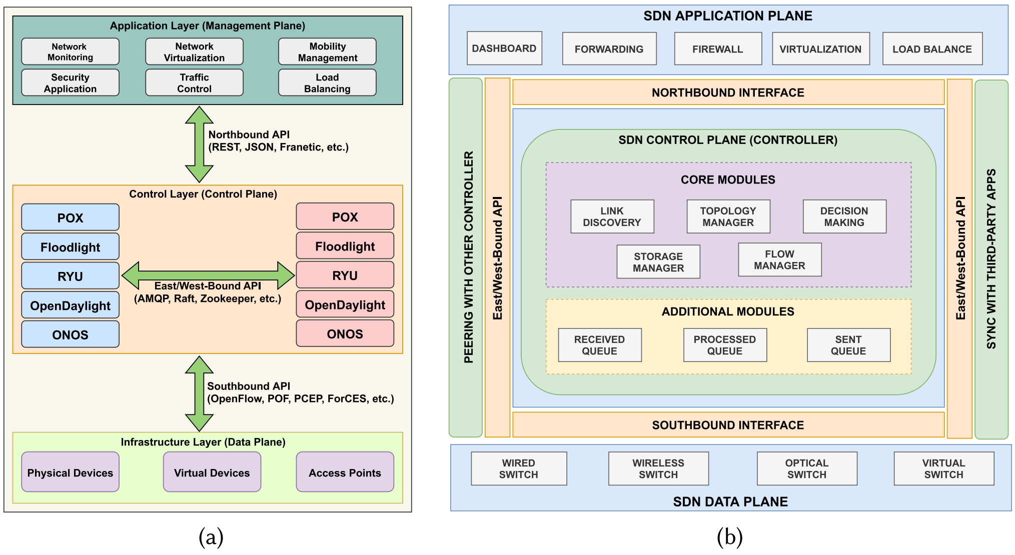

Fig. 1. SDN design. (a) The layered architecture. (b) Modular structure of a generic controller.

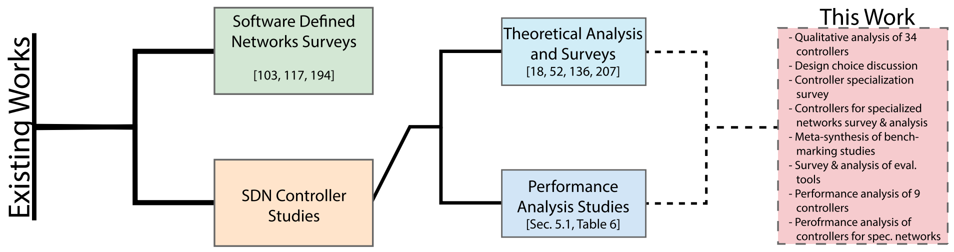
Existing Works
Software Defined Networks Surveys [103, 117, 194]
SDN Controller Studies
Theoretical Analysis and Surveys [18, 52, 136, 207]
Performance Analysis Studies [Sec. 5.1, Table 6]
This Work
- Qualitative analysis of 34 controllers
- Design choice discussion
- Controller specialization survey
- Controllers for specialized networks survey & analysis
- Meta-synthesis of benchmarking studies
- Survey & analysis of eval. tools
- Performance analysis of 9 controllers
- Perofrmance analysis of controllers for spec. networks

Fig. 2. Classification and scope of this article.

done before. Second, we present a survey of controllers specialized for different parameters and scenarios (e.g., IoT, blockchain, vehicular networks (VANETs)). Again, to the best of our knowledge, no other work provides such a detailed analysis. Third, we collect and analyze the exiting benchmarking efforts done by the research community for the lessons learned (Section 5 and Table 6). Most of these efforts are small scale and only analyze 2 to 4 controllers, whereas in this work we analyze 9 controllers, which are of interest to the community by three different benchmarking tools (Section 6). Furthermore, we also analyze the different tools and benchmarking metrics. To the best of our knowledge, there is no other work that brings all of these components together at the scale and magnitude shown in this work.

Contributions and organization. The specific contributions of this work are multi-fold:

- We present a brief introduction to SDN controller architecture and the need to benchmark.
- We present a qualitative comparative analysis of 34 different controllers for their properties including the major design choices, which affect the performance and capabilities of controllers.
- We present in detail and discuss the different use cases' specific enhancements for different controllers and their performance effects.

- We present a detailed comparative analysis and evaluation methods used for different controllers specifically designed (or optimized) for specialized networks such as the IoT, wireless sensor networks (WSNs), blockchain networks, and VANETs.
- We present a comprehensive study of benchmarking tools and the different techniques they employ for evaluating SDN controllers. This includes the discussion on existing works and their limitations/lessons learned, capabilities of benchmarking tools, and, most importantly, the details of metrics that should be used for quantitative evaluations.
- We conduct a quantitative analysis of 9 different controllers using three different benchmarking tools for a variety of metrics. The results presented show the actual performance of the controllers. We also present a comprehensive discussion on research findings not only for controller behavior but also for the metrics and tools used.

The rest of the article is organized as follows. Section 2 gives an overview of SDN controllers, followed by the classification, comparison, and design choices of controllers in Section 3. Specialized controllers are analyzed in Section 4, whereas Section 5 describes in detail the benchmarking studies, tools, and metrics. Quantitative evaluation results and research findings are detailed in Section 6. Section 7 concludes the article.

2 SDN CONTROLLER ARCHITECTURE

A controller also known as the Network Operating System (NOS) is the core component of any SDN infrastructure, as it has the global view of the entire network, including data plane SDN devices. It connects these resources with management applications and performs flow actions dictated by application policy among the devices. The proposals put forth for different controllers in the literature do not modify the underlying controller architecture but rather differ in terms of modules and capabilities. Hence, we find that presenting individual architectures is less useful for the reader. Here, we present the general architecture, as shown in Figure 1(b), and discuss its different modules.

Controller core. The core functions of the controller are mainly related to topology and traffic flow. The link discovery module regularly transmits inquiries on external ports utilizing *packet_out* messages. These inquiry messages return in the form of *packet_in* messages, which allow the controller to build the topology of the network. The topology manager maintains the topology itself that provides the decision-making module to find optimal paths between nodes of the network. The paths are built such that the different quality-of-service (QoS) policies or security policies can be forced during path installation. In addition, the controller may have a dedicated statistics collector/manager and queue manager for collecting performance information and management of different incoming and outgoing packet queues, respectively. The flow manager is one of the significant modules that directly interacts with data plane's flow entries and flow tables. It utilizes the Southbound Interface (SBI) for this purpose.

Interfaces. Different interfaces surround the core controller for interaction with other layers and devices [107]. SBI defines a set of processing rules that enable packet forwarding between forwarding devices and controllers. SBI helps the controller provision physical and virtual network devices intelligently. OpenFlow (OF) [119] is the most commonly used SBI and is a defacto standard for the industry. The fundamental responsibility of OF is to define flows and classify network traffic based on a pre-defined rule set. On the opposite end, the controller uses the Northbound Interface (NBI) to allow developers to integrate their applications with controller and data plane devices. Controllers support several northbound Application Programming Interfaces (APIs), but most of them are based on REST API. For inter-controller communication (hierarchical or flat) and communication with legacy routers, the East- and Westbound Interfaces (E/WBIs) are used. There is no standard communication interface for this purpose; hence, different controllers use different mechanisms. Moreover, heterogeneous controllers do not usually communicate with each other. Border Gateway Protocol (BGP) [149] is the most commonly used protocol for legacy router communication.

Modern SDN controllers and SDN design are not the first attempt at centralizing the network control. SoftRouter [105] and ForCES [198] separate control elements from forwarding elements but are limited to packet modification functionalities. Routing Control Platform [70] is an intra-AS platform to expand BGP. Path Computation Engine [66] enables clients to execute path computations in routers. Intelligent Route Service Control Point [54] introduces a path allocation module in an external router and provides dynamic connectivity features to enhance traffic flows throughout a network. The 4D project [76] was a theoretical clean-state solution to introduce a control plane for topology discovery and to provide traffic forwarding logic and rule sets. SANE [39] enables traffic forwarding and access control policies using a logically centralized server within enterprise networks. Similarly, Ethane [38] further improves the control/topology and pre-defined flow management.

The control plane of these earlier proposals was missing a broad range of matching header fields and also lacks a wide range of functionalities. As a result, SDN has become mainstream with the introduction of OF [119], which is a data plane API, and a robust centralized controller named *NOX* [77]. The majority of moderns SDN controllers are completely dependent on OF.

3 CONTROLLER CLASSIFICATION AND DESIGN CHOICES

To compare different SDN controllers, we have performed an extensive search of proposals not only in the academic literature but also in the commercial domain. Here, we first present the possible classification criteria of controllers, followed by the comparative analysis, and then different use case–specific enhancements.

3.1 Classification and Selection Criteria

The behavior of controllers is more or less the same across all of the proposals listed in Table 1. After analysis of 34 controllers, we conclude that the working, role, and responsibilities of the majority of these do not present any classification basis. Perhaps the only classification criteria that can be used is the *deployment architecture*. The initial aim of SDN was to centralize the control plane; hence, most of the controllers utilized a single controller. However, this created a single point of failure and scalability challenges. The distributed architecture allows usage of multiple controllers inside a domain, working in a flat or hierarchical formation.

In this work, we have not limited the selection of controllers to any specific criteria. Instead, we have collected all possible controllers from the literature and other documented projects. To the best of our knowledge, there is no other work that collects and compares such a large number of controllers.

3.2 Design Choices and Qualitative Comparison

Table 1 presents a comprehensive view of the different properties of the controllers. In the interest of space and the fact that not all proposals provide extensive details about their inner workings, we do not discuss each controller individually. Rather, we present the properties and design choices of controllers.

It is important to note that many of these controllers are not maintained, updated, or properly documented for the use of the research community. Similarly, many do not have their source code

Table 1. Comparative Analysis of SDN Controller Features

Ref.	Prog. Language	Arch.	NBI	SBI	EWBI	Platform	Interface	License	Multi-Threading	Modul-arity	Consis-tency	Docum-entation
Beacon [62]	Java	C	Ad hoc	OF 1.0	–	Linux, MacOS, Windows	CLI, WUI	GPL 2.0	✓	Fair	✗	Fair
Beehive [201]	Go	DH	REST	OF 1.0, 1.2	–	Linux	CLI	Apache 2.0	✓	Good	✓	Limited
DCFabric [46]	C, JavaScript	C	REST	OF 1.3	–	Linux	CLI, WUI	LGPL 3.0	✓	Good	✓	Fair
Disco [142]	Java	DF	REST	OF 1.0	AMQP	–	-	Proprietary	–	Good	✗	Limited
Faucet [16]	Python	C	–	OF 1.3	–	Linux	CLI, WUI	Apache 2.0	✓	–	✓	Good
Floodlight [11]	Java	C	REST, Java RPC, Quantum	OF 1.0, 1.3	–	Linux, MacOS, Windows	CLI, WUI	Apache 2.0	✓	Fair	✓	Good
FlowVisor [162]	C	C	JSON RPC	OF 1.0, 1.3	–	Linux	CLI	Proprietary	–	–	✗	Fair
HyperFlow [178]	C++	DF	–	OF 1.0	Publish & subscribe messages	–	–	Proprietary	✓	Fair	✗	Limited
Kandoo [200]	C, C++, Python	DH	Java RPC	OF 1.0-1.2	Messaging Channel	Linux	CLI	Proprietary	✓	High	✗	Limited
Loom [96]	Erlang	DF	JSON	OF 1.3-1.4	–	Linux	CLI	Apache 2.0	✓	Good	✗	Good
Maestro [37]	Java	C	Ad hoc	OF 1.0	–	Linux, MacOS, Windows	WUI	LGPL 2.1	✓	Fair	✗	Limited
McNettle [190]	Haskell	C	–	OF 1.0	–	Linux	CLI	Proprietary	✓	Good	✗	Limited
Meridian [17]	Java	C	REST	OF 1.0, 1.3	–	Cloud based	WUI	–	✓	Good	✗	Limited
Microflow [205]	C	C	Socket	OF 1.0-1.5	–	Linux	CLI, WUI	Apache 2.0	✓	–	✗	Limited
NodeFlow [25]	JavaScript	C	JSON	OF 1.0	–	Node.js	CLI	Cisco	–	–	✗	Limited
NOX-Verity [77, 177]	C++	C	Ad hoc	OF 1.0	–	Linux	CLI, WUI	GPL 3.0	Nox-MT [178]	Low	✗	Limited
Onix [102]	C++	DF	Onix API	OF 1.0, OVSDB	Zookeeper	–	–	Proprietary	✓	Good	✗	Limited
ONOS [24]	Java	DF	REST, Neutron	OF 1.0, 1.3	Raft	Linux, MacOS, Windows	CLI, WUI	Apache 2.0	✓	High	✓	Good
Open Contrail [65]	C, C++, Python	C	REST	BGP, XMPP	–	Linux	CLI, WUI	Apache 2.0	✓	High	✓	Good
ODL [113]	Java	DF	REST, RESTCONF, XMPP, NETCONF	OF 1.0, 1.3	Akka, Raft	Linux, MacOS, Windows	CLI, WUI	EPL 1.0	✓	High	✓	Good
OpenIRIS [108]	Java	DF	REST	OF 1.0-1.3	Custom Protocol	Linux	CLI, WUI	Apache 2.0	✓	Fair	✗	Limited
OpenMul [155]	C	C	REST	OF 1.0, 1.3, OVSDB, Netconf	–	Linux	CLI	GPL 2.0	✓	High	✗	Good
PANE [71]	Haskell	DF	PANE API	OF 1.0	Zookeeper	Linux, MacOS	CLI	BSD 3.0	–	Fair	✗	Fair
POF Controller [110]	Java	C	–	OF 1.0, POF-FIS	–	Linux	CLI, GUI	Apache 2.0	–	–	✗	Limited
POX [144]	Python	C	Ad hoc	OF 1.0	–	Linux, MacOS, Windows	CLI, GUI	Apache 2.0	✗	Low	✗	Limited
Ravel [191]	Python	C	Ad hoc	OF 1.0	–	Linux	CLI	Apache2.0	–	–	✓	Fair
Rosemary [164]	C	C	Ad hoc	OF 1.0, 1.3, XMPP	–	Linux	CLI	Proprietary	✓	Good	✗	Limited
RunOS [152]	C++	DF	REST	OF 1.3	Maple	Linux	CLI, WUI	Apache2.0	✓	High	✓	Fair
Ryu [153]	Python	C	REST	OF 1.0-1.5	–	Linux, MacOS	CLI	Apache 2.0	✓	Fair	✓	Good
SMaRtLight [33]	Java	DF	REST	OF 1.3	BFT-SMaRt	Linux	CLI	Proprietary	–	–	✗	Limited
TinySDN [53]	C	C	–	OF 1.0	–	Linux	CLI	BSD 3.0	✗	–	✗	Limited
Trema [171]	C, Ruby	C	Ad hoc	OF 1.0	–	Linux	CLI	GPL 2.0	–	Good	✗	Fair
Yanc [123]	C, C++	DF	REST	OF 1.0-1.3	Yanc File System	Linux	CLI	Proprietary	–	–	✗	Limited
ZeroSDN [101, 104]	C++	DF	REST	OF 1.0, 1.3	ZeroMQ	Linux	CLI, WUI	Apache 2.0	–	High	✓	Fair

C, centralized; CLI, Command Line Interface; DF, distributed flat; DH, distributed hierarchical; EWBI, east-west API, GUI, Graphical User Interface; WUI, Web UI; ✓, yes; ✗, no.

available anymore; hence, quantitative analysis is not possible (or even necessary). In the following, we discuss some of the core design choices that have been used in controllers listed in Table 1.

Programming language and implementation. Controllers have been written using different programming languages, such as C, C++, Java, JavaScript, Python, Ruby, Haskell, Go, and Erlang. In some cases, the entire controller is built using a single language, whereas for many other controllers, multiple languages are used in their core and modules. This amalgamation of different languages is usually done for efficient memory allocation, multi-platform compatibility, and, most importantly, achieving higher performance under certain conditions. Hence, the choice of language becomes an important factor in controller implementation. It is also important to note that the implementation details may also impact the performance of a controller. Therefore, the existence of bugs in code, the use of non-standard programming practices, and the maturity of the software become interesting evaluation criteria. Vizarreta et al. [189] show a comparison of ONOS and ODL controllers from the software implementation perspective and conclude that software reliability growth models can be used to increase the productivity of the overall system.

Architecture. The initial design decision of a controller is its architecture, which can be centralized or distributed. Centralized controllers are mostly used in small-scale networks, whereas distributed controllers can span across multiple domains. They can further be classified into flat, where all controller instances have equal responsibilities, or hierarchical, where a root controller is present.

Programmable interface (API). Generally, the northbound API (NBI) allows the controller to facilitate applications like topology monitoring, flow forwarding, network virtualization, load balancing, and intrusion detection based on the network events that are generated by data plane devices. However, low-level APIs like the southbound API (SBI) is responsible for enabling the communication between a controller and SDN-enabled switches or routers. Additionally, the east-west API is used by multiple controllers from different domains to form peering with each other in a distributed or hierarchical environment. Not all controllers provide all APIs, and only a select few have customized them for their specific use.

Platform and interface. These properties describe the implementation of the controller to be compatible with a specific operating system. The majority of controllers are built on top of Linux distributions. Moreover, to configure and view statistical information, some controllers provide graphical or web-based interfaces to the administrators.

Threading and modularity. A single-threaded controller is more suitable for lightweight SDN deployments. In contrast, multi-threaded controllers are suitable for commercial purposes such as 5G, SDN-WAN, and optical networks. However, a controller's modularity allows the integration of different applications and functionalities. High modularity allows a controller to perform faster task execution in a distributed environment. It is important to note that the worker threads are used for many monitoring tasks, which result in better performance. The basic task of flow installation itself is not hyper-threaded.

License, availability, and documentation. Most of the controllers discussed in this article are licensed as open source. However, a few have a proprietary license, which means they are only available through special requests or for research purposes. Regular maintenance of these controllers is also a challenging task for the developers, which is why a number of them do not receive regular updates. Nevertheless, the source code is available online, allowing anyone to make further changes according to the requirements. While accessing them online, we have found that the majority of them lack proper documentation. On the contrary, the ones that are updated regularly feature detailed and updated documentation for all of the available versions and also include community-based support.

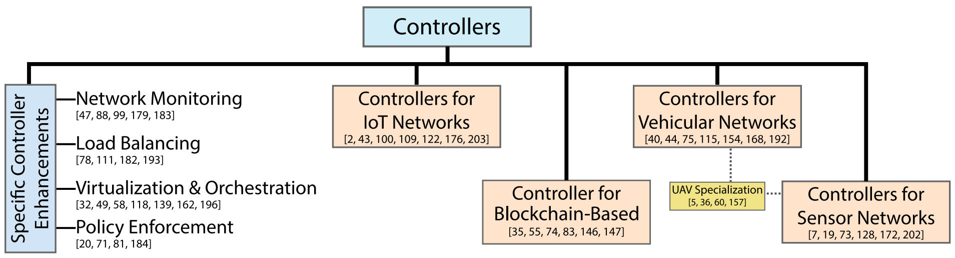

Fig. 3. Controller specializations.

4 COMPARATIVE ANALYSIS OF GENERAL AND SPECIALIZED CONTROLLERS

The discussion in the previous section and Table 1 give a general qualitative comparison. The majority of these controllers have been either discontinued (i.e., not maintained or updated) or have not been used by works in the SDN domain (i.e., academic research or industry). Hence, it is not useful to discuss each one in detail. In this section, we focus on more specific specializations of these controllers and then present detailed comparative analysis on them. Most controllers are designed for wired or optical domain networks, which are usually not mobile and have abundant resources. Some of the controllers have been optimized for specific enhancements. Similarly, there are scenarios where controllers are required to change or extend their architecture, applicability, and communication protocols to accommodate specialized networks. These networks include the IoT, VANETs, WSNs, and blockchain networks. Figure 3 shows these specializations and classification. In the following section, we first discuss these controllers (with specialized enhancements) and then discuss the controllers for specialized networks.

4.1 Use Case–Specific Enhancements to SDN Controllers

The adoption of different controllers and SDN, in general, has also triggered enhancements and use case–specific improvements for different controllers. Here, we have grouped these enhancements into different categories and summarize how they improve the capabilities of controllers.

4.1.1 Network Monitoring. Network monitoring has become one of the most crucial use cases of SDN controllers. The controller takes advantage of the global view of topology and proactively queries the performance. OpenTM [179] was proposed as a module for NOX, which is one of the earliest open source OF controllers. This monitoring scheme evaluates the traffic matrix of OF switches with a consistent polling rate. However, this also leads to higher monitoring overhead. Adrichem et al. [183] presented OpenNetMon, a Python-based module for the POX controller to monitor end-to-end per-flow QoS metrics like throughput, delay, and packet loss. From the statistical analysis results, the approach for monitoring throughput is excellent, although continuous polling of information may cause overhead on the controller. Flow monitoring is limited to edge switches only. Payless [47] implemented over the Floodlight controller is another query-based monitoring framework that can request the desired QoS metrics using a set of well-defined RESTful APIs. However, some trade-off between accuracy and overhead can lead to slight performance degradation for different polling intervals. SDN Interactive Manager [88] and OFMon [99] are two recent implementations of network monitoring modules that have been built over the Floodlight and ONOS controllers, respectively.

4.1.2 Load Balancing. The SDN controller plays a vital role in enabling load balancing in distributed systems by optimizing resource allocation, minimizing response time, and maximizing the throughput of that system. Without rewriting IP addresses, Handigol et al. [78] implemented a method where the NOX controller can be used along with the OF switch reactively to reduce response time for load balancing of multiple web servers. Contrarily, Uppal and Brandon [182] used address rewriting techniques for a NOX-based load balancer, which cuts down cost and brings flexibility. Another NOX-based proactive load balancer was proposed by Wang et al. [193], which uses OF wild card rules that can achieve faster adaptation with new load balancing weights and to redistribute the existing weight more efficiently. Based on switch migration technique, Liang et al. [111] presented a dynamic load balancing method that has been implemented over the cluster ODL controller. However, this method may fail in large-scale networks due to the coordinator node's recurring load collection issue.

4.1.3 Network Virtualization and Cloud Orchestration. With the addition of network virtualization techniques, SDNs have gained a new dimension with the ability to allow network slicing and multi-tenant hosting on existing physical network resources. FlowVisor [162] is the most popular SDN-based implementation to utilize virtual networks by leveraging OF functionality to abstract the underlying hardware. VeRTIGO [49] is an extension of FlowVisor that provides the controllers to choose the depth of virtual network abstraction required. This extension increases more flexibility in provisioning SDNs, although at the cost of hypervisor complexity. To reduce the complexity of network management, Xingtao et al. [196] presented an SDN controller built on Docker [57] to improve the deployment speed with expanded mobility. In the work of Drutskoy et al. [58], the flexibility of the NOX controller has been used as a container-based controller virtualization module to effectively cache and manage mappings between virtual networks and physical switches. HyperFlex [32] proposes a control plane virtualization model that mainly aims at achieving scalability, privacy, and extensibility. In this architecture, FlowVisor and Ryu controllers have been combined to provide the core hypervisor functions and to control the hypervisor network, respectively.

Cloud orchestration defines the integration of SDN controllers with a cloud-based resource manager, such as OpenStack [173], to enable dynamic inter-working between DCs, wide area networks, transport networks, and other enterprise networks. In the work of Mayoral et al. [118], ODL is integrated with OpenStack Havana [139] to evaluate the effectiveness of SDN in a cloud-based architecture where multiple DCs are located in different domains. In this architecture, the controller communicates with Havana using its REST NBI to perform critical tasks such as building, removal, and migration of virtual instances, which are located in inter-DC and intra-DC environments.

4.1.4 Policy Enforcement. To enhance security and flexible network management, an SDN controller can assign different policy-based decisions by implementing flow-based forwarding rules. Hinrichs et al. [81] implemented NOX as an application to provide access control and external authentication and to enable policy enforcement along with network isolation. PANE [71] presents an API to allow administrators to install policies for bandwidth allocation, access control, and path control. Additionally, the API provides the capability to query the state of the network or to provide information to the SDN controller regarding future traffic characteristics. PolicyCop [20], based on the Floodlight controller, is an autonomic QoS policy enforcement architecture that presents an interface for specifying QoS requirements in service-layer arguments and implementation through the OF API. In addition, it can monitor different policies so that control plane rules can be modified with changing traffic conditions autonomously. An extra module of the ONOS controller has been extended to implement a policy-based secure framework in the work of Varadharajan et al. [184]. The authors allowed end-to-end SDN services across various domains, including inter- and intradomain, using a wild card–based policy language, which includes a group of entities and services. Associated action such as acceptance or denial of a request is executed when a policy statement is satisfied.

Table 2. Comparative Analysis of SDN-Based IoT Controllers and Control Plane Architectures

Ref.	Control Plane	Controller Design*	Source Code & Documentation	Objective of Controller	Evaluation Method	Evaluation Tools	Benchmark Metrics	Remarks
[2]	C	Minor changes (to ODL)	Design diagrams only	IoT service mgt. Policy mgt. Multi-domain	Small-scale emulated 10 vSwitches & 4 FTP server	OpenStack Mitaka OVS	E-to-E latency. Traffic flow rate.	Better failure detection. Comparison to ODL only.
[43]	DH	New (IoT Dev. Mgt.) Minor changes (Net. Controller)	Source code [42] No doc.	Minimize flow-setup delay	Large-scale physical testbed 68 x Raspberry Pi 3B with Amazon Web Services	Docker OVS Cassandra GlusterFS	CPU & memory utilization. CPU temp. throughput. Response & converg. time.	Compared to Ryu, ONOS, ODL, Zero SDN, & BLAC. Different scenarios.
[100]	C	New [126, 127]	Source code & doc. [21, 125]	Node & state mgt. Security & policy conflicts	Small-scale emul. & physical testbed 2 x Odroid-XU32	CBench OVS 2.9.1 Mininet 2.2.1	Latency. Throughput. Policy overhead.	Partial eval. against NOX. Parallelized event-driven model performs better.
[109]	DF	Minor changes (general)	Theoretical only	Virtual & overlay device mgt.	Large-scale simu. & emulated 30 x IoT dev., 6 x virtual nets., & 4 x controllers	ORBIT sandbox Geni OMNet++	Delay. Throughput.	Framework evaluated against Devoflow, Hedera, & Geni. More controllers improve scalability.
[122]	C	Minor change (to POX)	Theoretical only	Task offloading	Medium-scale simulated 15 x APs, 5 x fog nodes	Mininet OpenNetMon	Task offloading & delay rate. Avg. hop count.	Evaluated against POX, which has no offloading
[175]	C	Minor changes (for IoT Protocol)	Design diagrams only	Control routing protocol. Monitor runtime components.	Medium-scale emulated 21 x IoT nodes	Cooja Contiki OS Zolertia Z1	Control overhead. Packet delivery ratio.	No benchmarking. Self-performance comparison.
[203] [204]	DF	New (IoT Dev. Comm.) Minor changes (ONOS & OSM)	Partial code & doc. [9]	Connectivity & flow-rule mgt. Security.	Small-scale emulated vCore servers, 6lowPAN devices, & Docker instances	Cooja Contiki OS Apache Kafka Django	Attack detection rate. Event processing rate.	Better performance shown in terms of enforcing necessary policies

C, centralized; DF, distributed flat; DH, distributed hierarchical.
*General: Changes are made to an abstract/generic controller model.

4.2 SDN Controllers for the IoT

An SDN controller in an IoT scenario is required to maintain simultaneous communication with the SDN switches (OF compatible access points) and the IoT devices [3, 69]. Additionally, the majority of IoT environments are heavily dependent on cloud and fog networks to provide optimal performance [6]. Therefore, IoT-aware SDN controllers would require an extended layer of SBI along with OF to communicate with IoT devices. Regarding the internal architecture of an SDN controller, additional modules like the protocol interpreter should be integrated with the main module. We present a comparative analysis of such solutions in Table 2 and a detailed discussion on performance in the following. As the focus of this work is on evaluation and benchmarking of controller performance, we discuss these details in more depth as compared to other SDN generic features.

Some of the works discussed in Table 2 (e.g., [43, 100]) have proposed completely new controllers. In contrast, others have made significant or minor modifications to existing controllers to work with IoT networks. Chattopadhyay et al. [43] propose a multi-layer control plane, where the top and middle layers contain service and network controllers, respectively. In contrast, the bottom layer has multiple lightweight controllers, termed as *micro-controllers* [42]. These micro-controllers are custom built on top of IoT devices and orchestrate with the network controller for information sharing and necessary task initialization. This controller as a service allows in-network processing that improves scalability and flow processing time. The performance has been evaluated using two large testbeds of multiple Raspberry Pi 3(B) nodes and the dockerized environment with 68 Amazon Web Service nodes, respectively. The authors compare their micro-controllers with ODL [113], ONOS [137], Ryu [153], and Zero [101, 104] using metrics like memory consumption, temperature, and utilization of the CPU. They also compare their framework with BLAC [84] using various IoT-based applications such as HTTP-server (data transfer), Cassandra (data-driven app), and Gluster (file sharing and fault tolerance). These tools are used to compare metrics such as throughput, response time, download time, and convergence time. Despite showing notable performance improvement for the framework and controller, the overall architecture is not technically detailed enough to be improved by other researchers. Moreover, the technicalities of integration between IoT-based devices and three identical types of controllers are not entirely defined. Kim et al. [100] also propose a new controller, but it is targeted for security policy enforcement and access control. The proposed SODA design interprets incoming raw packets from IoT devices and translates them into control messages, which later transforms as policy instructions for those devices. Latency comparison with existing NOX-MT controller and real-world testbed evaluations shows the efficiency and effectiveness of such hybrid controller adoption for IoT scenarios.

Some works [109, 175, 203] do not propose a new controller but have made significant changes to existing controllers to achieve better performance or to adapt them for specific IoT network functions. Zarca et al. [203, 204] present a unified SDN-NFV framework using SDN and an IoT controller for IoT device security. The IoT controller acts as an IoT gateway and enforces policies. The testbed implementation of this framework is staged with multiple machines, where ONOS [137] and OSM [64] are used for SDN security enforces. In contrast, the IoT controller uses Contiki OS 2.7 [59] with an Erbium CoAP [63] server and a policy-based repository and orchestrator services on an Intel Haswell processor-based workstation. Evaluation is entirely for security metrics such as incident handling efficiency, policy interpretation, and enforcement. This work presents a substantial contribution in terms of framework and testbed implementation. However, the authors only consider the self-defined evaluation and do not compare the performance with other solutions. Theadorou et al. [174, 175] propose heterogeneity and mobility-aware services in an IoT environment using a modified control and application plane. The controller uses adaptive-RPL [188] for device communication. The authors build a medium-scale emulated 6LoWPAN [85] testbed integrated with Cooja Simulator [176] and Zolertia Z1 [211] motes running Contiki OS [59]. The evaluation uses real trace data from Alay et al. [4] and measures the packet delivery rate. However, benchmarking is limited to the authors' self-defined experiments. The framework itself is promising for adoption into IoT-based networks but lacks necessary technical details, such as algorithms and packet processing methods for controller modules. Li et al. [109] propose LS-SDV, a double-layer distributed yet flat control plane to manage virtual networks and IoT devices in large-scale infrastructure. Physical and virtual controllers in two-layer overlay organizations manage real and virtual networks while focusing on flow scheduling and traffic optimization. Evaluation is done using OMNet++, as well as on a four-controller testbed. Thirty IoT devices use 180 video streaming traces to measure throughput and delay as compared to Hedera [1] and Devoflow [50]. Theoretically, this work provides complete algorithms and their proofs; however, lack of implementation details and source code makes it impossible for open benchmarking.

In other works (e.g., [2, 12, 14, 122, 129, 150]), minor changes have been proposed, such as the addition of simple modules at the application layer or on the top of the existing control plane. Most of these do not change the functionality of the controller but instead provide additional services. Misra and Saha [122] propose a dynamic task-offloading scheme using a centralized controller in a fog-enabled IoT environment. It uses ILP and greedy heuristic algorithms to find optimal processing fog nodes. The framework is implemented in an emulated testbed consisting of Mininet [121] and POX [144]. The evaluation is done against similar algorithms; hence, controllers' performance is not the measured parameter.

In some other works (e.g., [22, 150, 151, 206]), SDN has been used for IoT, but they do not propose a new control plane architecture or the controller. As the objective of this work is to present controller benchmarking, we have not included them in Table 2. For example, Ren et al. [151] present a distributed hierarchical control plane architecture to resolve the bottleneck issues with large-scale IoT devices and applications. However, technical details regarding the working of modules are not available, and evaluation is done using Matlab. Therefore, it is not possible to present or discuss implementable information that would be beneficial to the audience of this work.

Comparative analysis and insights. The design innovation of SODA [100] has a slight advantage over ANASTACIA [203], as SODA integrates an entirely custom-built Barista [126] controller. However, the ANASTACIA framework has a well-defined implementation, testbed specifications, and performance benchmarking criteria. Although both frameworks deploy NFV functionalities to detect and mitigate security vulnerabilities, evaluation metrics such as *detection time* and *incident handling performance* of ANASTACIA showcase its applicability. The rest of the works are limited to theoretical frameworks, and they only extend the functionalities of the existing controllers. Moreover, some of these works provide a limited comparison to similar proposals. Mininet and OVS are heavily involved in most testbeds along with existing controllers such as ONOS, ODL, Ryu, and POX. In the case of a simulated environment, Cooja Simulator is used along with Contiki OS.

Performance metrics used. The benchmarking metrics used in these works is very similar to those earlier described. However, they are much more simplified. These include *throughput*, *delay*, *control overhead*, *packet delivery rate*, and *end-to-end latency*.

4.3 SDN Controllers for Blockchain-Based Networks

Blockchain facilitates verification of transactions through distributed network authorization and then commits that data to an immutable ledger [28, 30]. The decentralization reduces control from a single node, eliminating the chance of single-point failure [27, 29]. As an emerging field for secure communication, blockchain has also been integrated with SDN [147], which has given numerous benefits, especially in the domain of IoT. In this section, we have collected and analyzed those works that are focused on blockchain and SDN amalgamation while keeping in view the controller enhancement and benchmarking as shown in Table 3.

StewARD [34, 35] presents a new SDN controller that allows the IoT device to interact with blockchain elements. The objective is to automate security risk detection in home-IoT devices. Although this work presented design details for grouping, network slicing, and communication, the testbed for evaluation is not present. Hence, the actual working or its benefits cannot be determined.

Rathore et al. [146] do not introduce an entirely new controller for the blockchain-based SDN paradigm; alternately, they make significant changes to the existing controllers. These changes are mainly to incorporate a blockchain layer parallel to the control layer, which contains different blockchain elements. From a benchmarking and testbed perspective, the work of Rathore et al. [146] proposes the use of multiple SDN controllers in the fog layer, which use a blockchain network to enforce security policies. The authors use a medium-scale testbed where the blockchain layer on the fog and cloud is implemented through the Ethereum [148] platform. Moreover, Mininet [121] is used to emulate the SDN switches. Additionally, Amazon EC2 is used as a cloud controller to implement machine learning functionalities. On the other side, the POX [144] con-

Table 3. Comparative Analysis of Controllers for SDN-Based Blockchain Networks

Ref.	Control Plane	Controller Design	Source Code & Documentation	Objective of Controller	Evaluation Method	Evaluation Tools	Benchmark Metrics	Remarks
[35]	DF	New controller (proposal only)	-	Trust level eval. Home IoT device failure reporting.	—	—	Perf. behavior. Trust assessment.	No benchmarking. Summary of implementation only.
[55]	C	Minor changes (to ONOS)	Design diagrams & algorithms only	Identify malicious instructions. Detect vulnerable flow rules. IDS.	Small-scale emulated	Mininet OVS OpenStack Multi-chain	False positive. Detection accuracy. Training efficiency.	Different attack scenarios evaluated
[74]	C	Minor changes (proposal only)	Diagrams & theoretical information	Manages RSU & gNodeB for 5G VANET. Security policies.	Medium-scale simulated WAVE, 1000-m Tx, 4 lanes, 10 cars/lane	MATLAB NS-3 Hyperledger Fabric 1.0.2	Packet delivery. Tx delay.	No control plane evaluation. Overall feasibility evaluated.
[83]	DH	Minor changes (new modules to Floodlight)	Algorithms only	Detect illegal flows. Security.	Small-scale emulated 1 x controller, 6 x OF switches, 1 x Ethereum node	Mininet OVS sFlow-RT Truffle Suite	Attack mitigation rate. Detection accuracy.	IDS & IPS as part of controller. REST API for BC interaction. Eval. for inter- & intra-domain.
[146]	DH	Major changes (to cloud EC2) Minor changes (to fog & edge POX)	Testbed implementation only	ML-based cloud controller. Detect & classify malicious patterns. Enforce security policies. Detect BC-based attacks.	Medium-scale emulated 15 x computers	Mininet OVS Ethereum Truffle Suite NSL-KDD dataset	Accuracy. Predictive value. Detection rate & time.	Self-defined test for eval. No comparison to existing controllers
[147]	C	Minor changes (for NFV & BC)	Design diagrams only	Aggregate network. Slice RF spectrum. Policy implementation.	Large-scale simulated 72 RF slices divided into BC, cellular, edge, & user layers	—	Throughput per user. RF allocation rate.	Testbed details not available. Self-defined RF allocation evaluation only.

BC, blockchain; C, centralized; DF, distributed flat; DH, distributed hierarchical.

troller is implemented on the fog layers, resulting in an interesting setup, as it has enabled the authors to measure the detection efficiency and resistance of the proposed decentralized framework. Traditional SDN properties are not presented in this work.

Several works [55, 83, 93, 143, 208] have presented minor changes by adding a module at the application layer for communication with an independent blockchain network. A distinct evaluation set up in this regard is by El Houda et al. [83], which uses a centralized SDN controller to mitigate DDoS attacks. The authors implement the framework in a small-scale emulated testbed. Mininet, OVS, and Floodlight [11] controllers are used to emulate multiple Linux-based hosts, OF switches, and a centralized SDN controller, respectively. The authors deploy their proposed smart contracts on a private blockchain using Ganache [181], an emulator to test blockchain-based smart contracts. Besides this, the smart contract is also initialized on Ropsten, an Ethereum-based testbed. sFlow-RT [86] is configured to work with the Floodlight controller to monitor the performance of the available switch ports. Evaluation is done based on the attack mitigation rate and detection accuracy. However, they are not benchmarked against other solutions. Other works (e.g., [74, 195]) also integrate blockchain with SDN. Nonetheless, they are not targeted for controller designs.

Comparative analysis and insights. In terms of implementation and measured efficiency, BlockSecIoTNet [146] provides a better overall framework. Components of the testbed scenarios are technically explained, and the work shows the implementation feasibility as well. On the contrary, the rest of the solutions listed earlier mostly emphasize theoretical representation and do not deliberate on the implementation and benchmarking of the solutions. Furthermore, the use of tools such as MATLAB or NS-3 does not demonstrate the working principle of blockchain-based smart contract applications, APIs, or communication protocol between SDN and blockchain. Most of the solutions use an emulated medium-scale testbed, where either Amazon EC2 or OpenStack is used to build a cloud-based network. In addition, Mininet with OVS is exploited to form the SDN scenario. The majority of the proposed works use ONOS or POX to implement necessary control plane functionalities as additional modules. For emulating a blockchain-based environment, Truffle Suite is used to deploy smart contracts, build the custom application, and initiate necessary tests. Perhaps the most important thing from an evaluation perspective is the lack of available implementation and for independent evaluation and analysis for future works.

Performance metrics used. Most of these solutions in this domain are designed to enforce security policies. Hence, the metrics for the controller and, more importantly, the complete framework evaluation have also changed. These include *attack detection accuracy*, *false-positive rate*, *attack mitigation ratio*, and *adaptation efficiency* regarding blockchain-based functionalities.

4.4 SDN Controllers for VANETs

In a software-defined vehicular environment, other than the management of forwarding devices in the data plane, the controller has to manage (and in some cases become) cellular base stations, the radio access network, Road Side Units (RSUs), and smart vehicles [41, 89, 199]. Additionally, recent 5G deployment integrates cloud, fog, and edge elements into the network. Therefore, distributed SDN controllers are necessary to tackle the challenges of current VANETs. In this section, we focus on the literature presented in Table 4, which aims at introducing new or improved controllers only (not general SDN or NFV solutions) for VANETs.

Some works [44, 154] propose significant changes to existing controllers. Chekired et al. [44] propose SliceScal, which is a hierarchical framework with three different controllers (i.e., edge, fog, and cloud controllers). Together they enable network slicing, mobility management, handover, and end-to-end communication for 5G VANETs. The authors use NS-3 [134] with Veins-SUMO [167] for vehicular traces and also build a testbed with ODL to manage OF switches and POX [144] as the cloud controller. The controllers are stand-alone machines, whereas Mininet 2.1.0 [121] and OF 1.3 [138] create the SDN environment. The setup used is relatively small, and the evaluation done is to measure delay/latency, where the comparison is not made with other state-of-art solutions. In another work, Sadio et al. [154] propose a prototype of a complete SD-VANET framework for cellular network infrastructure to facilitate intelligent forwarding decision making by the centralized controller. This controller communicates with various devices such as BS, RSU, and RAN elements. The work presents different algorithms and then evaluates them on a small testbed. It uses Zodiac Fx [133] switches and Ryu [153] as the centralized controller. The proposed routing scheme and the best path selection algorithm are implemented under the PureSDN [45] framework and later integrated to the Ryu controller. In contrast, the Odin [114] framework on TP-Link AC1750 devices creates the WiFi Access Points. Although the performance is not benchmarked against any other solution, the authors measure different metrics, including inter-channel handover and inter-technology handover. Moreover, the implementation is not available; instead, the algorithms are given in the article.

Some works (e.g., [13, 75, 115, 166]) have made minor additions to controllers to achieve better data sharing and provide authentication to devices. From an evaluation and testbed perspective, the works of Luo et al. [115] and Garg et al. [75] are interesting. Luo et al. [115] use OF-enabled RSUs, which use SBI to communicate with vehicles, whereas specialized NBI is used for applications. They use the OF extension [135], which is rarely used by the research community to evaluate SDNs. Garg et al. [75] introduce an SDN-based centralized control plane framework as a part of their proposed scheme to provide privacy and end-to-end security. It adopts the mobility model

Table 4. Comparative Analysis of Controllers for VANETs

Ref.	Control Plane	Controller Design*	Source Code & Documentation	Objective of Controller	Evaluation Method	Evaluation Tools	Benchmark Metrics	Remarks
[40]	C	Minor changes (general)	Design diagrams only	Path optimization. Load balance. Reduce control overhead.	Large-scale simulated 500 vehicles, 2 km^2, 20 x nodes	NS-3.26 SUMO v0.32 Open-StreetMap	Delivery ratio. Throughput. Delay. Routing overhead.	No benchmarking to controllers. Controller tasks given theoretically.
[44]	DH	Major changes (to POX & ODL)	Design diagrams & algorithms given	Resource sch.	Small-scale simulated 4 lanes on 20-m road, 10 smart vehicles	NS-3 Veins SUMO Mininet 2.1.0	Handling & propagation latency. Slicing rate. Delay.	Multi-controller arch. Evaluation for resource scheduling & slice mgt. Inter-controller comm. not defined.
[75]	C	Minor changes (general)	Design diagrams & algorithms given	IDS. Authentication.	Large-scale simulated 9 km^2 with highways, 300 vehicles	NS-3 SUMO SPAN	Detection time. False positives. Accuracy.	No benchmarking. Various attacks & IDS evaluated.
[115]	C	Minor changes (general)	Design diagrams only	Collect context on resources. Data sharing.	Small-scale simulated 1 x BS, 5 x RSUs, 7 vehicles	NS-3 SUMO	Received data ratio. Delay. Scheduling rate.	No benchmarking. Controller is a simulated BS.
[154]	DF	Major changes (to net. cont.) Minor changes (SDRAN cont.)	Algorithm given	Handover mgt. RAN mgt. Path selection.	Small-scale emulated 3 x APs, 4 OVS, 2 controllers	Zodiac FX PureSDN Wi-5 Project	Tx time. RTT. Handover rate. Latency.	Modules are integrated on Ryu & Odin-wi5 controllers. Eval. is of framework.
[168]	DH	Minor changes (general)	Design diagrams & algorithms given	Routing & path optimization	Large-scale simulated 20 x RSU, 1,500 km^2	NS-3 SUMO	State collection rate. Overhead. Latency.	Evaluation of routing only. No controller benchmarking.
[192]	DH	Major changes (net. cont.) Minor changes (for BS)	Design diagrams & algorithms given	Sense traffic patterns. Route comput.	Medium-scale simulated 6 x bus lines, 3 x RSUs, 15 x buses	Simulation tool in C++	Request lifetime. Vehicle interval rate.	Eval. limited to probabilistic analysis of routing strategies. Sim. tool details not given.

C, centralized; DF, distributed flat; DH, distributed hierarchical.
*General: Changes are made to an abstract/generic controller model.

from Sedjelmaci and Senouci [158], and two parallel highways with 300 vehicles are simulated using NS-3 and SUMO [170]. It compares the framework to other security solutions in terms of accuracy, detection rate, false-positive rate, and detection time.

Finally, some works (e.g., [40, 94, 132, 140, 168, 192]) present frameworks that target VANET and SDN hybrid systems. However, they either do not technically detail the controller design or control plane architectures. Despite this fact, a few of these are listed in Table 4 as they provide testbed specifications with some tools that would be beneficial to an interested reader.

Comparative analysis and insights. Most of the performance evaluations in SDN-VANET are initiated from a simulated testbed rather than real ones, due to implementation challenges of emulated vehicles or highways. As a result, the majority of the works in this section have multiple scenarios for both simulated and partially emulated nodes. In terms of emulated testbed implementation, The work in Sadio et al. [154] presents better technicalities and specifications, which involves multiple open source tools like PureSDN, Zodiac FX, Odin Agent, and the Python Twink [95] OF library. However, SliceScal [44] shows better performance when compared to other solutions. The implementation of SDN in VANETs is very promising in terms of tackling critical challenges like mobility, scalability, handover, and path selection. However, SDN alone cannot solve these issues. Therefore, integration of key technologies such as fog, cloud, NFV, network slicing, virtualization,

Table 5. Comparative Analysis of Controllers for WSNs

Ref.	Control Plane	Controller Design*	Source Code & Documentation	Objective of Controller	Evaluation Method	Evaluation Tools	Benchmark Metrics	Remarks
[7, 8, 73]	C	New set of controllers	Source code [120]	API for sensor nodes. Policy mgt. Trusted authority for nodes.	Large-scale emulated 100 x sensors, 80 km^2	Mininet IBM TPM	Received ratio. Delay.	ONOS extended for NFV & policy enforcement. No benchmarking. Basically an upgraded version of SDN-WISE.
[19]	DF	Minor changes (general)	Design diagrams & algorithms given	Optimize beam formation & energy allocation. Policies mgt.	Simulated	—	SNR. Energy req.	No benchmarking. Eval. of algorithms. Sim. specs. not given.
[128]	DH	Major changes (SDN-WISE)	Design diagrams only	Security & energy mgt. Cluster mgt.	Medium-scale simulated 36 nodes, grid topo.	Cooja Contiki Powertracek	Delivery rate. Delay. Overhead. Energy req.	Eval. against SDN-WISE & IT-SDN. WSN API implemented.
[172]	C	Minor changes (SDN-WISE)	Design diagrams & algorithms given	Cluster formation. Topo mgt. & routing.	Large-scale simulated 200 nodes	Cooja Contiki 3.0 SDN-WISE API	Loss rate. Overhead. Delay. Net. lifetime.	Eval. against SDN-WISE. NOS concept adopted from SDN-WISE.
[202]	C	Minor changes (general)	Design diagrams & algorithms given	Neighbor discovery & routing.	Medium-scale physical 6 nodes, 30-km^2 area	Raspberry Pi devices	Net. lifetime. Delivery rate. Energy req.	No benchmarking of controller. Incomplete testbed specs.

C, centralized; DF, distributed flat; DH, distributed hierarchical.
*General: Changes are made to an abstract/generic controller model.

and next-generation cellular networks (5G, 6G) will prove to be the decisive factor to improve scalability, reliability, and security in SDN-VANET frameworks. Another direction that is missing is that of unmanned aerial vehicles (UAVs); however, as it is more like a sensing application, hence we touch again on it in the next section of sensor networks.

Performance metrics used. The overall evaluation metrics used in SDN-based VANET solutions are *end-to-end delay*, *handover ratio*, *packet overhead rate*, *delivery ratio*, *round-trip time* (RTT), and *handling latency*.

4.5 SDN Controllers for WSNs

Preserving the available energy to increase the lifetime of the network is considered to be one of the most critical metrics for evaluating the performance of WSNs [124]. Therefore, the controller in the software-defined WSN plays a significant role as it reallocates the resource-hungry tasks from the sensor nodes. Furthermore, the controller may also be responsible for managing and configuring nodes (e.g., the transmission range) to optimize the network overhead. The controller also makes a significant impact in terms of security and policy management. Table 5 presents a comparative analysis of the controller and control plane architectures, which have contributed significantly in this regard.

The most comprehensive work in WSNs is that of Galluccio et al. [73] (SDN-WISE), which was further augmented by Anadiotis et al. [7, 8]. It proposes an entirely new controller as an integrated component of the WSN framework. Anadiotis et al. [7] have further built a flexible flow-based rule framework to improve energy efficiency in WSNs. The most compelling aspect of this framework is a lightweight NOS (similar to Contiki OS [59] or RIoT [61]), with different modules for path management, network-function consolidation, context distribution, resource storage, and rule identification. In addition, the authors propose new NBI and SBI to communicate with sensor nodes in the data plane. Furthermore, it provides sensor-based applicability to the ONOS controller. Performance evaluation and benchmarking are done for controller response time, payload efficiency, and RTT in different WSN network settings.

Other works (e.g., [19, 128, 172, 197, 202]) have added minor modules in their work to enable node cluster formation and improve QoS, energy management, and optimization. Notable among these works are those of Tan et al. [172], who propose a QoS-based routing protocol and a clustering algorithm for workload reduction, and Younus et al. [202], who present an energy-aware routing protocol with a centralized SDN controller. Ndiaye et al. [128] proposed multi-layered hierarchical control plane architecture to allocate resource-oriented task distribution management modules for wireless sensor environment. The work of Tan et al. [172] uses Cooja Simulator [176] with Contiki 3.0 [59] to model a small wireless network and evaluate the performance in terms of network lifetime, end-to-end delay, and control message overhead with other algorithms. It is interesting to note that this work relies on SDN-WISE,; hence it shows a slight improvement in comparison to it. The work of Younus et al. [202] is evaluated on a physical testbed consisting of multiple Raspberry Pi [145] devices like sensor nodes and a separate desktop PC as the controller module. Evaluation metrics such as network lifetime, packet delivery ratio, number of an existing node, average delay, and energy efficiency are considered to benchmark the framework against the AODV [141] protocol.

Comparative analysis and insights. The domain of WSNs has been extensively researched, and IoT architectures have steadily replaced most of its implementations. Hence, there are very few recent articles on controller designs for it. The work of Galluccio et al. [73] is perhaps the only solution that effectively uses SDN controllers in a wireless environment. Most of the evaluation of SDN in WSN is based on simulation using Cooja and Contiki OS. In this regard, other works [7, 73] are better solutions as they extend the ONOS [137] controller, which can then be integrated with Mininet. However, a significant challenge is of benchmarking in the WSN paradigm. As not many solutions are available, and the generic controller does not have wireless capabilities, there is nothing to compare to. Other works have more or less emphasis on the theoretical analysis of the proposed solutions rather than demonstrating as practical implementation. Exceptional use of a software-defined WSN can also be found in UAVs, as they are also wireless and energy-constrained devices. Very few works on the SDN-UAV domain are available [5, 36, 60, 157]; hence, specialized distributed controllers for them can be a primary research direction. Most of these only address routing and network management but not detail the controller architecture.

Performance metrics used. The overall evaluation metrics used in SDN-based WSN solutions are *packet delivery*, *energy consumption*, *average delay*, *network lifetime*, and *control message overhead rate*.

5 BENCHMARKING METRICS AND TOOLS

Theoretical comparison based on features and properties does not reflect the actual performance of any controller. Hence, real deployment and benchmarking are necessary for accurate evaluation. Here, we first present an overview of the necessity and importance of evaluating controllers, and then discuss existing efforts for benchmarking along with valuable lessons learned. Finally, we present a list of performance metrics, which should be used in benchmarking of controllers.

Evaluating or benchmarking the performance of a controller can be done either through simulation/emulation or by using a hardware-based testbed. Although hardware testbeds provide measurements that are closer to actual values in a production environment, their cost is significant for the research community. Hence, emulation-based evaluations are common practice. Nevertheless, for benchmarking of SDN controllers, the software tool used has to be extremely efficient and precise. In this section, we present some well-known tools available for benchmarking, followed by analysis of their properties and benchmarking capabilities.

5.1 Existing Works and Lessons Learned

Prior to this article, numerous works [15, 26, 51, 87, 91, 92, 97, 98, 116, 131, 156, 159, 160, 169, 180, 209] used multiple techniques, tools, and testbeds to evaluate the performance of several SDN controllers, including scalability, reliability, efficiency, and robustness.

In Table 6, we compile most of the existing works associated with the evaluation of the controller performance and the notable findings. The majority of these works [97, 116, 156, 159, 160, 180, 209] use CBench [163] to evaluate the performance based on latency and throughput. In most cases, throughput mainly correlates with the threading capability of a controller; regarding the number of flows, it can process in a specified time slot. Some other works [91, 92, 131, 160] extend CBench to integrate support with the operating system's kernel and compilers like Java and Python. The aim is to improve the threading scalability of a controller regarding the system's I/O modules. Some works [87, 169] include simulation-based environments where hosts and vSwitches are virtualized to evaluate the impact of topology on the performance of a controller. In these experiments, the load balancing functionality is extensively tested. Moreover, the work of Shalimov et al. [160] evaluates the reliability of the controller by generating vulnerable flows. Energy consumption has been evaluated in other works [92, 116] using fat-tree or DC topologies. In the following, we give a brief description of some of the notable works.

Tootoonchian et al. [180] present the CBench [163] tool for the evaluation of different controllers. They perform multiple flow-based experiments using it to compare the effectiveness and performance of NOX-MT, a multi-threaded adoption of the NOX controller with other controllers like NOX, Beacon, and Maestro. Despite showing a notable improvement in performance, NOX-MT fails to identify some of the limitations of NOX, such as massive utilization of dynamic memory allocation and redundant representation of multiple requests.

Shah et al. [159] compare four multi-threaded controllers (NOX-MT, Floodlight, Beacon, and Maestro) for architectural features like multi-core availability, controller impact on the OF switch, packet batching, and task processing. The authors use CBench to compare these controllers based on their throughput and latency performance. Beacon shows better performance in these two scenarios due to its ability to use the multi-core and multi-threading functionalities. In addition, the dynamic changing of packet sizes allows Maestro to perform better in the latency test.

Shalimov et al. [160] present a framework named *HCprobe* to compare seven different SDN controllers: NOX, POX, Floodlight, Beacon, Ryu, MUL, and Maestro. To compare the effectiveness of these controllers, the authors performed some additional measurements like scalability, reliability, and security, along with latency and throughput. The testbed analysis presents some security vulnerabilities along with the reliability issues with MUL and Maestro controllers. However, Beacon, MUL, and Floodlight obtained minimum latency, whereas Beacon performed relatively well in the throughput test. The analytic hierarchy process method is used by Khondoker et al. [98] to analyze POX, Floodlight, ODL, Ryu, and Trema based on many standards like virtual switch support, modularity, documentation, programming language compatibility, and availability of the user interface. According to their calculation, Ryu was elected to be the most suitable controller based on these properties. However, the analytic hierarchy process method is subjective and may lead to different outcomes in different scenarios. Zhao et al. [209] use multi-core and many-core testbeds to evaluate NOX, Maestro, Floodlight, and Beacon on the aspect of multi-core utilization efficiency, performance scalability, and energy consumption regarding DC environments. The work emphasizes existing controllers' limitations in taking advantage of the concurrency in modern hardware. In the work of Salman et al. [156], the performance of well-known centralized and distributed SDN

Table 6. Comparative Analysis of Different Benchmarking Studies

Ref.	Testbed Specifications	Evaluation Tool Used	Controller(s) Evaluated	Evaluation Metrics	Optimization Objectives	Lessons Learned
[15]	3 testbeds, 3 servers with Xeon E5-2.40 GHz, 32 GB of RAM. 2 NICs, 1 Gbps each.	CBench, Mininet	ONOS, ODL	Throughput, topology discovery time, inter-controller traffic, network downtime	Not specified	Regarding topology discovery time, ODL takes 8x & 15x less than ONOS for 63 & 127 vSwitch, respectively. In a single node, ONOS's network downtime remains lower with 1 second as compared to ODL's 5 seconds.
[26]	Not specified	Mininet, Open vSwitch, Traffic Generator	POX & Floodlight	Round-trip delay. Average throughput.	Not specified	Simple controllers are better suited for configuration-related tasks. Feature-based controllers are good for performance-based tasks.
[51]	1 testbed, 3 servers with Xeon E5-2.30 GHz, 256 GB of RAM. 2 NICs, 1 & 10 Gbps.	CBench, YourKit	ONOS, ODL	Latency, throughput, memory usage, CPU utilization, thread scalability	Hyper-threading. One-to-one mapping on vCPU & pCPU.	More than 8 vSwitches improve ONOS's throughput as opposed to ODL. Task-batching improves latency for multi-threaded controllers. ONOS's throughput decreases 13.5% in VM versus a physical setup.
[87]	1 × quad-core, 1 × one octa-core testbed	Mininet, Open & Indigo vSwitch	POX	CPU utilization, topology impact, ping delay	Not specified	Number of switches impacts the flow installation time. Mininet utilizes maximum system memory. Initial ping delay is larger than avg.
[91]	Single testbed with 4 servers (dual core). 100-Mbps link speed.	OFCBenchmark	NOX, Floodlight, Maestro	Round-trip time. Send & response rate. Packet processing rate.	Implementation boost libraries to handle threads	Transmitting larger flows helps in detecting congestion in networks.
[92]	Not specified	OFCProbe	NOX & Floodlight	Impact of fat-tree topo. Load balancing.	Java library used to handle OF connections	Topology impacts the flow processing time. Efficient handling of switch depends on the characteristic of the controller.
[97]	5 × server with core i5 CPU	CBench	Floodlight & ODL	Throughput, latency, failure	Not specified	Custom profile is proposed for CBench. Controllers may suffer from memory leakages.
[98]	Not specified	Analytic hierarchy process	POX, Floodlight, ODL, Ryu, Trema	vSwitch support, modularity, docum., API compatibility.	Not specified	Evaluation method is subjective. Testing process may affect the outcome.
[116]	1 × multi-Core, 1 × many-core testbed. 10-Gbps link speed.	CBench	NOX-MT, Floodlight, Beacon, Maestro	Latency, throughput, energy consumption, I/O threading impact	Floodlight learning switch. CBench delay parameter. Maestro config file modification.	Number of switches & cores impacts NOX-MT's performance. CPU types & system architecture impact scalability.
[131]	2 Xeon testbeds	OFCProbe	ONOS	Topology discovery time, path provision time, ASYN. msg. process time	Not specified	Number of links has equal impact as number of switches regarding performance. Reactive path provisioning time relies on length of the corresponding path.
[156]	Single testbed with octa-core CPU. 10-Gbps link speed.	CBench	NOX, POX, Floodlight, ODL, ONOS, Ryu, IRIS, Beacon, Maestro	Latency, throughput	Not specified	Controller's SBI allows additional support for future Internet architecture
[159]	1 × cluster with 2 separate Xeon servers. 8 Gbps link speed.	CBench	NOX-MT, Beacon, Maestro, Floodlight	Throughput, latency, threading scalability, delay sensitivity	Switch partitioning. Packet batching. Task batching.	Switch partitioning & switch batching impact throughput. Packet batching & task batching impact delay sensitivity.
[160]	2 separate Xeon servers. 10-Gbps link speed.	CBench Hcprobe	NOX, POX, Floodlight, Ryu, Mul, Beacon, Maestro	Throughput, latency, reliability, security	Flow modification. Customized workload.	Scalability of controller depends on the number of cores. Not every controllers can handle a heavy workload.
[169]	Dual-core virtual testbed	Open vSwitch, Cluster Testbed, HTTP Generator, REST Client	ODL, ONOS	Flow installation rate. Flow reading rate. Failover time.	Controllers customized for the WAN environment	Size of a cluster has impact on the flow installation rate. Failover time of a controller depends on the number of devices. Latency has significant impact on large-scale WAN.
[180]	1 × quad-core & 1 × octa-core server. 2-Gbps link speed.	CBench	NOX, NOX-MT, Beacon, Maestro	Throughput, latency	Batching I/O. Boost Async I/O.	Number of switches impact the controller performance
[209]	Single testbed with quad-core Xeon server	CBench, Open vSwitch	NOX, POX, Floodlight, Ryu, Beacon	Throughput, latency, threading capability	Python interpreter, Hyper-threading.	HT offers performance improvement for Java-based controllers. Reliability, trustworthiness, usability, & scalability should be considered equally.

controllers is studied using CBench. The results show that both MUL and Libfluid MSG (written in C) achieved the highest throughput under an increasing number of switches, whereas Python-based Ryu and POX obtained better scores in the latency mode. However, with the increasing number of threads, both Beacon and MUL performed better, whereas Python-based controllers failed to show satisfying performance.

Two of the most widely implemented and adopted controllers(ONOS and ODL) are evaluated against each other in the work of Bah et al. [15] regarding adaptation with the rapid changes in the existed topology. The authors deployed single and cluster-oriented testbed scenarios using GEANT topology to conduct experiments using the Mininet [121] emulation tool. ODL achieves a better performance ratio in both of these environments in terms of detecting any possible changes in the topology compared to ONOS. ONOS outperforms ODL due to precise backup and recovery modules on the underlying controller architecture in terms of any obstacles or failure in the network. Using the same two controllers mentioned earlier, Darianian et al. [51] did performance evaluation in both physical and virtual scenarios in terms of latency, throughput, and thread scalability using CBench. OpenStack Kilo [173] is deployed to build the virtualized testing environment. Based on the evaluation of the physical and virtual environment, ONOS outperforms ODL in terms of latency and throughput due to ONOS's better adaptability with thread and socket processing with the provided hardware. The focus of this study is more on the threading and processing capabilities of the controllers.

5.2 Benchmarking Tools

The following are some of the commonly used tools for benchmarking. Table 7 provides a comparative analysis of the three main tools used for evaluation in this work.

CBench [163] is one of the fundamental benchmarking tools with an open source license. It is designed explicitly for evaluating the performance of OF SDN controllers, which support OF 1.0 and 1.3. However, due to compatibility limitations, controllers with OF 1.3 may experience performance issues. There are two basic evaluation metrics in CBench: latency and throughput. Regarding the latency measurement, the vSwitch forwards a single *packet_in* message toward the controller and waits for a response. Tests can be repeated several times to obtain the average performance. The total number of acknowledgments obtained in a test period is used to compute the average latency. As for throughput measurement, each vSwitch continuously sends as many *packet_in* messages as possible to estimate the capability of the controller.

HCprobe [160] is an open source extension of CBench, developed with the combination of Python and Shell scripts, to provide additional performance evaluation capabilities, such as reliability and scalability. The emulated switch can send vulnerable OF messages to controllers to check for resiliency and trustability. In addition, the test engine utilizes a Linux kernel, which allows customizable and scalable tuning of CPU threading. As a result, the tester can obtain more accurate performance statistics of an SDN controller.

Hvbench [48, 165] is an open source benchmarking tool designed for benchmarking hypervisors in a distributed SDN-based infrastructure. This tool can be used as container-based instances for multiple SDN controllers and data plane devices. The availability of various workload scenarios allows Hvbench to achieve horizontal and scalable performance evaluations of the SDN hypervisors. The tool has a few limitations. First, it relies upon the initialization of hypervisor. Therefore, it is not designed for physical SDN infrastructure. Second, the evaluation process depends on the pre-configured distribution of inter-arrival OF traffic. As a result, the tool does not support other customized or modified packet types for traffic flows.

OFCBenchmark [91] is built using C++ and Boost libraries to address some of the limitations of CBench. The components of this benchmarking tool include a graphical dashboard (built with

Table 7. Comparison of Benchmarking Tools

Tool	Advantages	Limitations	Built on	License	Availability	User Interface
CBench [163]	Faster analysis execution. Platform independent. Source code is available.	vSwitches limited to 256. Supports only OF 1.0. Flow length is limited. Supports only IP-based traffic.	C	Open source	Yes	CLI
HCprobe [56, 160]	Haskell-based build allows API for custom benchmarking. Initiate malformed OF messages to test vulnerability detection. Workload stress testing for controllers.	More like an update to CBench only in terms of features. Compatibility issues with recent controllers (ONOS, ODL).	Haskell	Open source	Yes	CLI
Hvbench [48, 165]	Supports multi-tenancy. Container-based instances. Lower CPU consumption. Pre-defined distribution. Dynamic workload scenarios.	Functional for hypervisor-based environment only	C++	Open source	Yes	CLI Web UI
OFCbenchmark [91]	Load-balancing evaluation. Container-based instances. Lower CPU consumption. Visual stats representation. Specific packet auditing.	Lacks support for ONOS, ODL, & recent controllers	C++ & Delphi	Proprietary	On request	CLI GUI
OFCProbe [72, 92]	Hassle-free communication between controller & testing system. Immediate request processing. Best-effort distribution. Cross-platform support. Well documented.	Heavy memory & CPU usage due to implementation details	Java	Open source	Yes	CLI
OFBench [112]	Designed for data plane benchmarking. Supports generic hardware. Automated test engine based on controller-agent.	Does not support control plane benchmarking. Accuracy depends on controller's stability.	N/A	Proprietary	Yes	CLI Web UI
OFNet [161]	In-depth performance analysis. Self-defined topology. Various traffic profiles. Flow event syntax.	Traffic generator & benchmarks rely on topology. Slower test duration.	C	Proprietary	On request	GUI
Perfbench [10, 31]	Emulates large OF flows. Works on virtualized & non-virtualized scenarios. Pre-defined & custom distribution. High-throughput workload support (i.e., DCs). Control & data plane benchmarking.	Virtualization may lead to control plane interference	C++	Open source	Yes	CLI
PktBlaster [186]	1,000 emulated switches. Customized switch groups. Detailed statistical results. Accuracy is better than CBench.	No customized topology. No application-based traffic. Free edition lacks deep analysis.	N/A	Proprietary	Onrequest	Web UI
WCbench [67, 68]	Automated evaluation process. Behaves like CBench wrapper. Easy on CPU & memory. Virtualized instances can be executed through Vagrant [80].	Built on the top of CBench. Only supports specific ODL versions. Depends on SSH connection.	Python	Open source	Yes	CLI

Delphi) and a virtualized scalable vSwitch, which is the core module and includes a client that can administer evaluation tests. The tool offers distributed benchmarking by allowing clients to run in multiple instances and offers extensible benchmarking such as RTT, flow installation rate, and CPU utilization.

OFCProbe [72] is an upgraded version of OFCBenchmark, which concentrates on maximizing the flexibility of SDN controllers by emulating a significant amount of OF switches in a large-scale environment. It is redesigned using Java to make it a platform-independent tool and also to overcome the virtualization overhead caused by an SDN emulation tool like Mininet [121]. The tool analyzes the impact of the network topology during the evaluation executed by the client component.

OFNet [161] is a combined approach to integrate OF network emulation with performance monitoring and visual debugging of SDN controllers. OFNet can be deployed in a system to generate different types of topologies. The inbuilt traffic generator produces different types of network traffic. It is capable of measuring performance characteristics of the controller, such as flow generations, flow failures, CPU utilization, flow table entries, average RTT, and the latency of flow setup.

Perfbench [10, 31] is another performance evaluation tool that is developed using the libfluid C++ library [187] to initiate precise, scalable, and high-performance-oriented evaluation for both virtualized and non-virtualized OF-dependent SDN operations. In addition, this tool can be exploited for both controller-based or switch-based performance evaluation in a multi-tenant hypervisor environment. The versatile feature sets allow Perfbench to support compatibility for both OF 1.0 and 1.3 versions and can generate multiple types of OF packets such as *packet_in*, *packet_out*, *echo_request*, *echo_reply*, *feature_request*, *feature_reply*, *flow_mode*, and *port_stats* messages.

PktBlaster [186] is a unified test solution that emulates large-scale SDN networks, including network infrastructure and orchestration layers of SDN controllers. The free version with limited capabilities offers features such as latency and throughput measurement with different testing profiles (i.e., TCP, UDP, ARP_Request, and ARP_Reply.) A throughput test determines the rate at which the controller configures the flows in the switches. The latency test gives the exact time in milliseconds, which the controller takes to process flow in the switch. Although the free version is limited to 16 switches and 64 MAC addresses, it offers additional properties like flow tables, group tables, meter tables, size of the switch buffer, and maximum entries per flow table.

WCBench [67, 68] is another variant of CBench written in Python and utilizes the core library module of CBench. Compared to CBench, the feature set of this tool goes beyond latency and throughput and offers additional aspects of automated evaluation with detailed and graphical statistics. Although it extends the support of OF to version 1.3, the compatibility of WCBench is still limited to specific versions of the ODL controller.

5.3 Benchmarking Metrics and Their Impact

In this section, we present a detailed list of performance metrics that can be used to benchmark SDN controllers. Table 8 outlines the grouping and description of each of these metrics. Some of these have also been identified by Vengainathan et al. [185]. However, we have extended this list and grouped them to eliminate the confusion regarding terminology. Generic terms, such as throughput and latency, can have significantly different meanings depending on the measurement process. Additionally, there can be other metrics to evaluate a controller, such as security and reliability. However, we refer to them as non-measurable parameters that are more subjective. We leave their classification as future work. The measurable parameters are grouped as follows.

Throughput metrics. Throughput is usually measured as a rate for processing flow requests by the controller. The vital thing to note is that it is not the flow installation time (path provisioning). From the test tools perspective, it is the number of *packet_in* messages sent, and the corresponding *packet_out* messages received per unit time. These requests could be synchronous or asynchronously coming from the vSwitches in a real environment. As a result, we consider parameters such as the processing rate of asynchronous and synchronous messages along with transmission rate for send and response messages to determine the throughput of the controller.

Latency metrics. This group of metrics is measured in time units. Similar to throughput, it only deals with the time between packets sent to controller and response received at the vSwitch. Many

Table 8. Classification of Benchmarking Metrics and Tool Capabilities

Measurable Metrics		Description	Benchmarking Tools		
Group	Parameters		CBench	PktBlaster	OFNet
Throughput	Async message processingrate	Determines number of flow requests a controller can process per unit time. A processed request does not mean a successfully installed flow.	✓	✓	❍
	Sync message processing rate		✓	✓	❍
	Send & response rate		×	❍	✓
Latency	Async message processing time	Denotes the delay or time duration between request from the vSwitch & response received back	✓	✓	✓
	Sync message processing time		✓	✓	✓
	Round-trip time		×	❍	✓
Flow related	Path provision time (proactive/reactive)	Determines efficiency of a controller to install flows, or measures that include communication between a source & destination	×	✓	✓
	Path provision rate (proactive/reactive)		×	✓	❍
	Flow reading rate		×	×	❍
	Flow installation time		✓	✓	✓
	Load balancing		×	❍	×
Topology	Topology discovery time/size	Measures SBI performance, detects topology-type, & monitors changes on the topology	×	✓	✓
	Topology change time		×	✓	✓
Threading	Thread capability	Indicates the utilization efficiency of a controller regarding the OS & physical hardware resources	✓	✓	✓
	I/O impact		✓	✓	×
	Control session capability		×	❍	×
	vSwitch CPU utilization		×	×	✓
Others	Forwarding table capacity	Miscellaneous parameters that can be measured for specific scenarios	×	✓	×
	Ping delay time		×	×	×
	Energy consumption		×	×	×
	Network provision time		×	×	×
	Controller failover time		×	❍	×

factors can affect the latency of a controller, including computation time required by the controller and link delay. Therefore, parameters like RTT and processing duration for asynchronous and synchronous messages are evaluated under latency experiments.

Flow-related metrics. These metrics deal with the complete path provisioning and flow installation. The primary difference between this and throughput is the complete path. Throughput only measures the rate from vSwitch to the controller and back to vSwitch. However, complete flow installation requires the installation of flow entries at other vSwitches along the path. We acknowledge multiple parameters in this group, such as provision rate and time for installing either proactive or reactive path, flow interpretation rate, and overall load balancing capability of the controller.

Topology -based metrics. The ability to detect or determine a topology including its type (single, linear, overlay, and tree), size, and the number of integrated nodes altogether represents a vital aspect of evaluating the efficiency of a controller. Interaction with its SBI also plays a significant role in these metrics. As a result, topology-based performance benchmarking includes the size of the topology and detection time regarding multiple variants of testbed topologies.

Threading and session metrics. This set of metrics identifies controller competence for utilizing the system architecture, hardware capabilities, and I/O units. Optimization of thread-based capabilities like multi-threading offers several advantages of task batching, event scheduling, process flows as groups, and, most importantly, increases the controller's flow processing time and rate.

Parameters such as CPU usage for running vSwitch instances, I/O batch processing, thread capabilities of the provided hardware, and control session efficiency are taken into consideration in this category.

Miscellaneous metrics. Here we group other parameters that can also be used for evaluating the controllers. Some of these can be crucial in specialized scenarios, such as energy consumption in mobile environments where controllers are deployed on energy-constrained devices. Similarly, in situations where hardware failure is a concern, the failover time needs to be reduced so that backup controllers can take over as quickly as possible. Therefore, in this portion, we group parameters such as controller failover time, energy consumption, ping delay time, forwarding table capacity, and network provisioning time. A controller with some knowledge of these metrics (or the network administrator) can lead to a more optimized managed network.

6 QUANTITATIVE EVALUATION AND BENCHMARKING OF CONTROLLERS

This section discusses the performance of 9 different controllers using previously described benchmarking tools. To the best of our knowledge, no previous work has compared such a large number of controllers and performed cross comparison using different tools for different metrics. The controllers evaluated are Nox-Verity, POX, Floodlight, ODL, ONOS, Ryu, OpenMUL, Beacon, and Maestro. The reason to select these out of previously discussed 34 controllers is (1) availability of controller source code, (2) compatibility with current Linux distributions, (3) interface-ability with the benchmarking tools, and (4) interest of the community. By observing the controllers used for specialized networks in earlier sections, it can be deduced that many of the new proposals are based on the controllers mentioned previously. Hence, providing a baseline for comparison will be beneficial to the community.

6.1 Evaluation Setup

The evaluation has been done in a partially emulated environment. The controller runs on a native Linux 16.04.03 LTS system, with an i5-9400F 2.9-GHz processor, 16 GB of DDR4 RAM, and solid-state NVMe m.2 interface storage. The tools and virtualized testbed environment run on a separate 3.4-GHz i7-6700 processor with 16 GB of DDR4 RAM and SATA interface solid-state memory. Note that the virtual environment has 12 GB of RAM allocation and two of the cores from the processor. The systems are linked with 1-Gbps links through a dedicated gigabit switch, whereas the virtual links in the virtualized domain are also 1 Gbps. Note that the capacity of links is not a bottleneck in the evaluation process.

In this work, we use three of the tools (i.e., CBench, PktBlaster, and OFNet) to evaluate different controllers. It is important to note that none of the tools available can measure all performance statistics and do not measure the same metric in the same manner. In most of the previous works and the output of tools, the metrics are somewhat simplified. For example, the throughput of a controller can be interpreted in several ways. Similarly, as shown in Table 8, the latency can be determined using different metrics. The columns on the right side of the table show each metric that can be directly measured, indirectly measured, or not measurable by a specific tool. Hence, we have tried to the best possible extent to make them similar. However, once the parameters are set, all controllers use the same values, as shown in Table 9. It is important to note that all tools and controllers are running on the same platform and hardware; hence, there is no platform bias. Furthermore, the reproducibility of the results has been ensured by testing the VMs on different hardware, whereas the operating system has been kept the same. We have observed similar trends for the results in these experiments.

Table 9. General Evaluation Parameters

Parameter	Specifications
Number of switches	2–128
Link speed	1 Gbps
Number of worker threads	4 (for controller)
Number of available CPU cores	4 (for controller)
Number of test iterations (confidence interval)	20 (Avg. results are plotted)
Number of unique hosts perswitch	64
Duration between each iteration	5 seconds (where applicable)
Single test duration	300 seconds
Traffic profile	UDP (TCP for PktBlaster)
Flow measured by	Packet-In, Packet-Out, Flow-Mod
Packet length	∼ 64 bytes
Flow transmission rate	250 flows per second
Topology type	Fat-tree

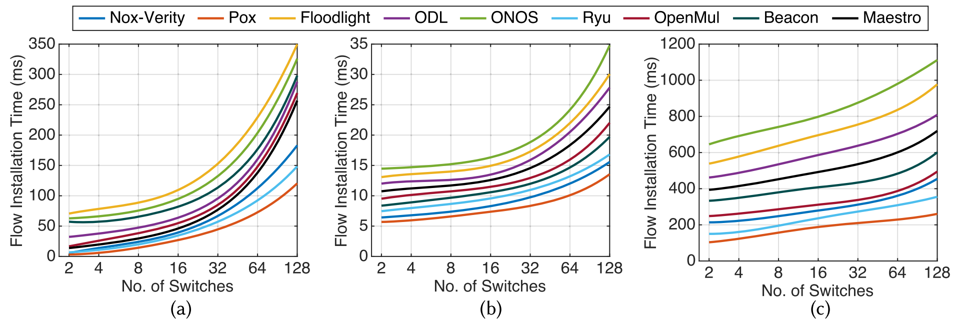

Fig. 4. Latency performance with varying number of switches. (a) CBench. (b) PktBlaster. (c) OFNet.

6.2 Latency Performance

The latency performance of a controller is generally measured as the time required for flow installation. More precisely, it is the time between the *packet_in* message sent and the *packet_out* message received at the switch. Figure 4 depicts the measured latency by each of the tools, for the nine different controllers evaluated, against the increasing number of switches in the topology.

In Figure 4(a), it can be observed that the performance of all controllers follows a similar trend; however, at a higher number of switches, the increase in latency for Nox-Verity, POX, and Ryu is not as steep as the rest of the controllers. It is important to note that the performance is relative to the hardware setup; hence, on more high-capacity systems, the performance will be better. The rapid increase in installation time is experienced around the 16–32 switches mark. Here, Nox-Verity, POX, and Ryu experience an 160% increase between 32 and 128 switches, whereas the rest experience an approximate 180% increase. Figure 4(b) shows the flow installation time for PktBlaster, and we see a similar trend as in the case of CBench. The increase is more drastic around the 32 switches mark. ONOS, Floodlight, and ODL tend to have higher (approximately 5–15 ms higher) installation times as compared to other controllers. In the third evaluation in Figure 4(c), we show the flow measurements from OFNet. It is important to highlight that the direct results from OFNet are reported against time and hence requires individual simulations for the different

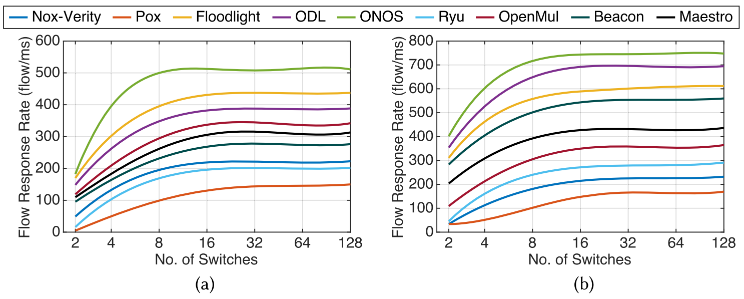

Fig. 5. Throughput performance measured against number of switches. (a) CBench. (b) PktBlaster.

number of switches. It can be observed that the performance in terms of better flow installation is of POX, Ryu, and Nox-Verity, whereas ONOS, Floodlight, and ODL take significantly more time in installing the flows (in order of 100 s of milliseconds). Here the difference is approximately 190% between the two groups. However, the increase does not reflect any drastic change as the number of switches increases.

One of the contributions of this article is to demonstrate the difference in outcome for the same metric under similar emulation and test environments for different tools. As can be seen from Figure 4, the *y*-axis scale varies extensively for all three tools. For CBench, the measured latency is in the orders 100 s of milliseconds but capping at 350 ms in the given scenarios, whereas in PktBlaster, the same controllers are observed to perform in 10 s of milliseconds, and in total contrast, the evaluation for OFNet is much higher. This difference can be attributed to the way measurements are done, the load on the controller, and the implementation of the controller. The measurement and metrics have been discussed in the earlier sections and in Tables 7 and 8. The load on all controllers is modeled to be identical, but as described earlier, the different tools generate this load in different ways. CBench is more sequential in generating the packets, whereas PktBlaster is parallel in nature. Hence, the actual load on the controller may vary. Similarly, the controller coding and implementation may also introduce processing delays with different degrees; however, it is difficult to dissect the coding of the tools or the controllers to pinpoint the heavy processing points. The second cross-analysis observation is the growth factor of different controller times. In Figure 4(a) and (b), we observe that the time increases exponentially, whereas in the evaluation of OFNet, this increase is smooth.

6.3 Throughput Performance

The throughput performance metric is measured using CBench and PktBlaster only, as shown in Figure 5. OFNet does not provide a direct measurement of flow processing. However, an indirect estimation can be done through sent and received OF messages, which are discussed in a separate section. Similar to the latency evaluation, this metric is also evaluated against the increasing number of switches in the topology.

In throughput mode, CBench switches send multiple packets at once without waiting for individual replies (opposite to the latency calculation mechanics). Figure 5(a) shows the results obtained from CBench. It can be observed that Nox-Verity, POX, and Ryu remain the lowest performers, whereas controllers like ODL, Floodlight, and ONOS perform significantly better. The difference is approximately 142% on average between these two groups. The general observation of a rapid increase in throughput is due to the number of hosts connected to each switch (which is modeled through the number of flows per second created at each switch as shown in Table 9). Hence, the traffic flow requests quickly amplify to a point after which no significant increase in

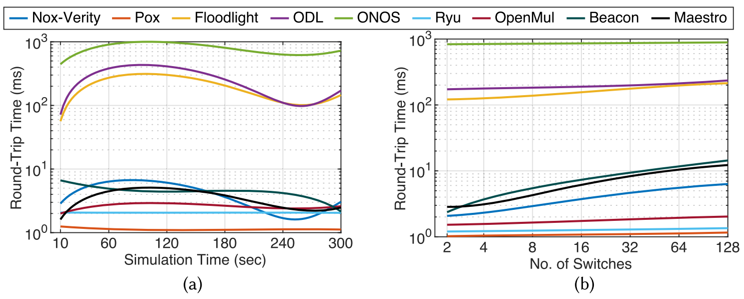

Fig. 6. RTT by OFNet. (a) Avgerage RTT against time (8 switches). (b) Average RTT against switches.

throughput is observed given the testbed setup. In Figure 5(b), the performance of PktBlaster can be observed. The performance of Floodlight, ODL, and ONOS is the best among all controllers compared, whereas Nox-Verity and POX are at the lower end. Here the difference is approximately 250% among these two groups. The increase in the number of switches increases the *flow_in* messages, and the throughout lines become flat around the 8 switches mark on average.

Similar to the earlier analysis, the tools differ in the throughput metric as well; however, the change is not too drastic. All controllers tend to perform better in PktBlaster evaluations as compared to CBench, in the order of 10 s of flows per second. Specifically, ODL and Floodlight show significant gains in performance. The peak performance in PktBlaster is approximately 200 flows per second higher on average for the top contenders. However, this should not be interpreted as an improvement in controller performance but rather is a measurement artifact of the benchmarking tool. The important point to note is that comparing results across different tools is not a good analysis practice. At the same time, one fact that does stand out is the commonality in the flattening of curve at the 8 switches mark, which is observed in both graphs in Figure 5.

6.4 OFNet Specific Measurements

In this set of experiments, we focus specifically on the performance metrics offered by OFNet. It is important to note that this work only focuses on the metrics reported directly by the benchmarking tools. Although OFNet reports several metrics, in the interest of space we only show the RTT and OF messages sent/received.

6.4.1 Average RTT. RTT evaluation is a crucial factor to consider when identifying the location of controller deployment. It identifies the communication delay between the controller and the switch. If the controller and switches are physically far apart, the increased RTT will contribute to increased latency. Similarly, the time complexity of packet processing at the controller affects the overall performance. Note that the RTT measurements in OFNet are not flow-installation time. We have run multiple experiments to determine the average RTT against a different number of switches in the topology, and we also show the results against time for 8 switches specifically as a sample. Remember that OFNet reports the measurements against time, and average calculations have to be done manually to plot against the number of switches as shown in Figure 6 (log scale). Based on the tree topology, Figure 6(a) shows that ONOS has high RTT in the range of 408 to 985 ms; however, it remains stable for the duration of the simulation. Yet Ryu and OpenMUL (minimum 2.04 ms and maximum 3.19 ms) have the least RTTs, mostly because of less complex algorithms involved at the controller. However, less complexity does not translate to better complexity; instead, they may be attributed to fewer controller capabilities. In Figure 6(b), the RTT is plotted against the number of switches. Note that the scale is in a log; hence, the slope of the line

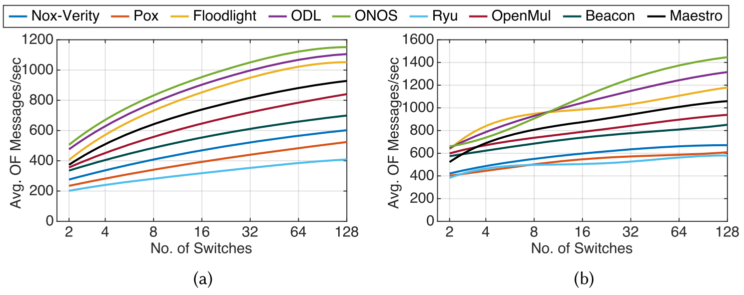

Fig. 7. OFNet Flow measurements. (a) Sent OF messages by switches. (b) Received OF messages by switches.

in some controller cases shows a significant increase in RTT. An interesting observation here is that RTT is not the flow installation time; hence, the increase in RTT is most probably due to the topology and load on the links, with some contribution in the form of controller processing.

6.4.2 OF Messages Sent and Received. OFNet measures several parameters, out of which these two are of significant importance, as it indicates the amount of control overhead required to maintain the flow entries and the flows themselves. It can also be used as a crude indicator of flows per second. The number of OF messages sent to the controller are loosely the *packet_in* messages by the switches for path establishment, whereas the received OF messages from the controller at switches are *packet_out* and *packet_mod* messages. In Figure 7, we measure these against the increasing number of switches in the topology. Intuitively, it can be observed that the increase in the number of switches also increases the number of OF messages generated by them, whereas the number of received messages is higher than the requests. However, the smoothness of lines is due to the high confidence interval as described earlier. In Figure 7(a), we observe that the sent OF messages are higher for ODL, ONOS, and Floodlight as compared to the rest of the controllers. The difference is, on average, 402 messages per second. We also observe a minor tapering off at 128 switches for these three controllers. We have not found any anomaly in the data and therefore attribute it to the physical limitations of the testbed. This is also corroborated with the fact that beyond 128 switches, the testbed does exhibit abnormal behavior, which is the main reason that all experiments have been limited to 128 switches. In Figure 7(b), we observe that more (approximately 25%) messages are sent back from the controller to switches in the case of ODL, ONOS, and Floodlight. As compared to the Nox-Verity, POX, and Ryu group, it is three times the rate at 128 switches. Note that these are per-second averages of messages.

6.5 Performance Analysis of Specialized Networks

In this section, we present a comparative analysis of the performance of different controllers (and control plane solutions) for specialized networks (i.e., IoT, WSNs, and VANETs). Several points should be noted in this quantitative performance analysis. First, a detailed comparative analysis of these controllers has already been discussed in the article (Sections 4.2, 4.4, and 4.5); hence, here we only focus on the quantitative performance. Second, this section does not implement the solutions, as most of them do not have any public code available. Therefore, we only elaborate on those works that have provided the implementation information to some extent,. Third, we have not discussed any of the blockchain solutions (Section 4.3) here, as none of the proposed systems have implementations available.

Controller/control plane for IoT. It can be observed in Table 2 that only some works [43, 100, 203, 204] have given details of implementation; hence, only these solutions are discussed here. Note that the discussion is based on the results provided by the source articles, as replication of the

Table 10. Quantitative Performance Analysis of Controllers/Control Planes for Specialized Networks

Ref.	Solution	Testbed	Performance Metrics (only relevant ones reported here)	Remarks
IoT Networks (Reference Table 2)				
[43]	SDC on micro-controllers	HW V	**CPU Utilization:** *Ryu*: Same requirement as without micro-controllers (maximum 27%). *ODL*: Majority (70%) of controllers need 17% CPU; rest go up to 50% CPU. *Zero*: Majority (60%) required <5% CPU, whereas max is <30%. **Memory Consumption:** *Ryu & Zero*: Similar (between 200 MB and <900 MB). *ODL*: Majority (60%) require 600 MB to 1 GB. **Flow Setup Time (Ping, RTT):** *Aloe*: Best case (200–500 ms); worst case (300–700 ms). *BLAC*: Best case (110–300 ms); worst case (580–900 ms). Link failure determines the best or worst case.	ONOS failed due to memory requirement. Zero [101] works better for Aloe.
[100]	Security framework for IoT	HW E	**Throughput:** 100% increase in throughput against NOX-MT (164.9K vs. 79K req./second). **Latency:** 34.4% reduction in latency against NOX-MT (0.8 vs. 1.22 ms).	Performance attributed to highly parallel event handler
[203] [204]	Security framework for IoT	HW V	No comparison to other controllers. Self-comparison shows that it utilizes 100% CPU for its policy operations.	Minor assessments shown
Vehicular Networks (Reference Table 4)				
[44]	SDN core for 5G VANET	S	**Avg. End-to-End Delay:** Increasing distributed controllers show a 20% to 40% delay reduction (ms). **Request Handling Latency:** With SDN-based 5G slicing, the request handling latency drops to <1 ms, even with an increasing number of requests.	Primary comparison against non-SDN system
[154]	SDN-based VANET	HW E	**Tx Speed and Delay (AP to Controller):** Suggests that concurrent APs decreases Tx speed in backbone (order of 10s of Megabits per sec), but the RTT/delay is not affected.	Minor self-benchmarking
[192]	Delay-constrained routing in VANET	S	**Delivery Ratio:** No cross comparison to other solutions. **Bandwidth Efficiency:** No cross comparison to other solutions.	Self-assessment with different routing strategies
Wireless Sensor Networks (Reference Table 5)				
[7, 8, 73]	SDN-WISE SD-WISE	HW	**RTT (Self Micro-Benchmark):** RTT increases by approx. 50% between 3 and 5 hops for different topologies. Increasing payload (by 10 bytes) increases RTT marginally. **Efficiency—Control to Payload (Self Micro-Benchmark):** Higher payloads increase efficiency (from 25% to 65% efficiency between 10- and 100-byte payloads).	SDN-WISE is a major solution in WSN
[128]	Mgt. services	HW	**Packet Delivery Rate and Delay:** Compared to IT-SDN*: 4% to 8% gain. **Energy and Computation:** Compared to SDN-WISE: <1% improvement shown. **Topology Variation:** Compared to SDN-WISE 95% reduction in control packets, and 10x reduction in delay (ms) is claimed.	Solution is built on top of IT-SDN,* which is built on SD-WSN [7]
[172]	Clustering Topology mgt.	S	**Control Message Overhead:** Claims improvement, but unit of ‘J’ is not clear. **Packet Loss Rate:** Compared to SDN-WISE 1% improvement in packet loss rate. **End-to-Endd Delay:** Depending on QoS better and worse against SDN-WISE.	No significant improvement against SDN-WISE observed

E, emulation; HW, hardware based; S, simulation; V, virtual testbed.
*No verifiable reference available.

results has not been possible. Chattopadhyay et al. [43] present a software-defined service (SDC) for IoT devices (Aloe), which is tested on hardware and then in a virtual environment. The SDC requires distributed micro-controllers in the topology, and the authors present interesting results by comparing different controllers, as shown in the Table 10. The CPU utilization of Zero controller [101, 104] was far less than the ODL, which is intuitive given the complexity and capabilities of ODL. However, their results also show that Ryu's CPU requirement was similar to having no micro-controllers, which is odd. They also show several other metrics, but we present the flow setup time, which is measured using Ping; hence, it is basic RTT. This is compared to another SDC called *BLAC* [84]. Kim et al. [100] present a security framework with a custom-built controller based on Brista [126, 127] and show throughput and latency against NOX-MT. Given the hardware setup, the proposal has a 100% increase in throughput and 34.4% less latency. Cumulatively, these works also claim to be better than the majority of controllers (except ODL) for IoT needs. The work of Zarca et al. [203, 204] performs no comparison to any controller and also shows *itself* utilizing 100% CPU.

Controllers for VANETs. For VANETs, the solutions presented in other works [44, 154, 192] describe some algorithms and implementation details as part of the work. Hence, in Table 10, we summarize their performance. It is interesting to note that although these solutions modify controllers, none of them actually presents a performance from a controller's perspective. Most of these are minor self-assessments, without any comparative analysis to other solutions.

Controllers for WSNs. WSNs have been researched for many years now and have significantly matured. The integration of SDN into WSN is also not new, but still there is a major lack of controllers in this hybrid domain. SDN-WISE [73] is perhaps the only contender as a controller, whereas its comparison to other lightweight controllers is still missing. In Table 10, we provide two other solutions [128, 172] that present implementation details. Similar to the problem identified in VANETs, cross comparisons are very limited. Ndiaye et al. [128] show a comparison to SDN-WISE, but no significant improvement in performance is observed.

7 RESEARCH FINDINGS AND CONCLUSION

Analyzing and benchmarking the performance of a controller is a challenging task. This work presents a systematic and comprehensive analysis of general and specialized controllers. We first qualitatively compare 34 controllers and then discuss different design choices, where multi-threading and distributed nature of controllers are the most effective ones. There are two main findings. First, the majority of the controllers proposed in the literature have no implementation available, and the details available are not sufficient for the third person to code it. Hence, other than theoretical comparison, it is not possible to evaluate them. Second, some of the controllers that do have public implementations available are not maintained or are very rudimentary in nature. For example, some works use initial (years old) implementations of POX, NOX, and ONOS, which inherit the limitations of older code (software aging problems), thus impacting the reliability (and repeatability) of the performance shown.

Controllers have been specialized in two ways. One is to improve specific functions of the controller, such as monitoring, orchestration, and load balancing, whereas the second is to specialize it for a specific network/use case. We present an analysis from both perspectives. First, the majority of modern networks such as IoT and VANETS still utilize the same controller structures that were developed for wired infrastructure. Very few new models have been designed from scratch that are targeted for the specialized nature of the network. Second, the availability of implementations of these designs is scarce, which is a major challenge in evaluation and enhancement. Although many of these claim to have been tested on physical networks, without precise implementation details, their usage, repeatability, and enhancement is not possible.

This work also discusses in detail the existing benchmarking studies, the lessons learned, and the metrics used for evaluation. Based on this, we have categorized and defined several benchmarking metrics. An important factor in performance analysis is the tool used for evaluation; hence, we have identified 10 different tools and comprehensively compared their capabilities. First, we noticed that some of the available features of tools, such as packet length and vSwitch buffer size, impact the performance of the controller. However, it is important to note that the outputs given by any tool also indicate the performance of components used in complete topology. Isolating the performance of the controller from the results is not possible. Second, the coding imperfections of tools and measurement processes have to be considered while comparing any solutions. The evaluations presented in research works that do not use any specific benchmarking tool cannot be corroborated with evaluation using a benchmarking tool. This is mostly in those cases where the parameters/metrics are improperly defined. For example, RTT (via ping) is presented as controller latency. Third, utilization of benchmarking tools like OFNet allows the definition of custom topologies with a variety of traffic profiles. Hence, picking the correct tool and topology is as important as defining metrics to be measured.

Finally, this work presents a quantitative evaluation of nine controllers using three different tools for several metrics. Based on the qualitative and quantitative analysis, we determine that some controllers, such as ODL and ONOS, have better flow installation rates at the cost of higher latency. First, considering latency and throughput, multi-threaded controllers, including centralized ones (Floodlight, OpenMul, Beacon, Maestro) and distributed ones (ODL and ONOS), perform significantly better than centralized and single-threaded controllers like POX and Ryu. However, they also require more physical resources to perform efficiently. Moreover, a single-threaded centralized controller can still perform better in simplified topologies, whereas multi-threaded controllers are more suitable for complex environments. Second, placement of the controller in physical topology directly impacts several performance parameters. This work has not explored this direction; however, topology-specific controller placement experiments would be interesting future work, especially for specialized networks. Third, quantitative performance evaluation of specialized network controllers is a big challenge. With the current lack of publicly available code for these solutions, these works are neither repeatable/reproducible nor they can be enhanced further by the research community.

Although there are no clear winners in wired, wireless, general, or specialized networks, the final message is to ensure that the modern controller design has to be specialized for the target network and must be evaluated and an appropriate tool designed for the specific environment with the best possible implementation of the controller itself. This holds specifically for drone and aerial vehicular networks.

REFERENCES

[1] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'10)*. 281–296.

[2] Saba Al-Rubaye, Ekhlas Kadhum, Qiang Ni, and Alagan Anpalagan. 2019. Industrial Internet of Things driven by SDN platform for smart grid resiliency. *IEEE Internet of Things Journal* 6, 1 (Feb. 2019), 267–277.

[3] Iqbal Alam, Kashif Sharif, Fan Li, Zohaib Latif, M. M. Karim, Sujit Biswas, Boubakr Nour, and Yu Wang. 2020. A survey of network virtualization techniques for Internet of Things using SDN and NFV. *ACM Computing Surveys* 53, 2 (July 2020), Article 35, 40 pages.

[4] Ozgu Alay, Andra Lutu, Rafael Garcia, Miguel Peon-Quiros, Vincenzo Mancuso, Thomas Hirsch, Tobias Dely, et al. 2016. Measuring and assessing mobile broadband networks with MONROE. In *Proceedings of the IEEE 17th International Symposium on a World of Wireless, Mobile, and Multimedia Networks (WoWMoM'16)*. IEEE, Los Alamitos, CA.

[5] Mohannad Alharthi, Abd-Elhamid M. Taha, and Hossam S. Hassanein. 2019. An architecture for software defined drone networks. In *Proceedings of the IEEE International Conference on Communications (ICC'19)*. IEEE, Los Alamitos, CA.

[6] Ricardo S. Alonso, Inés Sittón-Candanedo, Sara Rodríguez-González, Óscar García, and Javier Prieto. 2019. A survey on software-defined networks and edge computing over IoT. In *Proceedings of the International Conference on Practical Applications of Agents and Multi-Agent Systems*. 289–301.

[7] Angelos-Christos Anadiotis, Laura Galluccio, Sebastiano Milardo, Giacomo Morabito, and Sergio Palazzo. 2019. SD-WISE: A software-defined WIreless SEnsor network. *Computer Networks* 159 (Aug. 2019), 84–95.

[8] Angelos-Christos G. Anadiotis, Laura Galluccio, Sebastiano Milardo, Giacomo Morabito, and Sergio Palazzo. 2015. Towards a software-defined network operating system for the IoT. In *Proceedings of the IEEE 2nd World Forum on IoT*. 579–584.

[9] ANASTACIA Consortium. 2019. ANASTACIA: Advanced Networked Agents for Security and Trust Assessment in CPS/IOT Architectures. Retrieved May 14, 2020 from http://www.anastacia-h2020.eu/.

[10] Andreas Blenk. 2018. Perfbench Github Repository. Retrieved May 21, 2020 from https://github.com/tum-lkn/perfbench.

[11] Atlassian. 2016. Floodlight Controller. Retrieved May 12, 2020 from https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343542/Getting+Started.

[12] Luigi Atzori, Jose Bellido, Raffaele Bolla, Giacomo Genovese, Antonio Iera, Antonio Jara, C. Lombardo, and G. Morabito. 2019. SDN&NFV contribution to IoT objects virtualization. *Computer Networks* 149 (Feb. 2019), 200–212.

[13] Gagangeet Singh Aujla, Neeraj Kumar, Sahil Garg, Kuljeet Kaur, and Rajiv Ranjan. 2019. EDCSuS: Sustainable edge data centers as a service in SDN-enabled vehicular environment. *IEEE Transactions on Sustainable Computing*. Epub Ahead of Print. March 25, 2019.

[14] Michael Baddeley, Usman Raza, Aleksandar Stanoev, George Oikonomou, Reza Nejabati, Mahesh Sooriyabandara, and Dimitra Simeonidou. 2019. Atomic-SDN: Is synchronous flooding the solution to software-defined networking in IoT? *IEEE Access* 7 (2019), 96019–96034.

[15] Mamadou T. Bah, A. Azzouni, M. T. Nguyen, and Guy Pujolle. 2019. Topology discovery performance evaluation of OpenDaylight and ONOS controllers. In *Proceedings of the 2019 Conference on Innovation in Clouds, Internet, and Networks.* IEEE, Los Alamitos, CA, 285–291.

[16] Josh Bailey and Stephen Stuart. 2016. Faucet: Deploying SDN in the enterprise. *Communications of the ACM* 60, 1 (Dec. 2016), 45–49.

[17] Mohammad Banikazemi, David Olshefski, Anees Shaikh, John Tracey, and Guohui Wang. 2013. Meridian: An SDN platform for cloud network services. *IEEE Communications Magazine* 51, 2 (Feb. 2013), 120–127.

[18] Fetia Bannour, Sami Souihi, and Abdelhamid Mellouk. 2018. Distributed SDN control: Survey, taxonomy, and challenges. *IEEE Communications Surveys & Tutorials* 20, 1 (2018), 333–354.

[19] Xuecai Bao, Hao Liang, Yuan Liu, and Fenghui Zhang. 2019. A stochastic game approach for collaborative beamforming in SDN-based energy harvesting wireless sensor networks. *IEEE Internet of Things Journal* 6, 6 (Dec. 2019), 9583–9595.

[20] Md. Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, and Raouf Boutaba. 2013. PolicyCop: An autonomic QoS policy enforcement framework for software defined networks. In *Proceedings of the IEEE SDN for Future Networks and Services (SDN4FNS'13).* IEEE, Los Alamitos, CA, 1–7.

[21] GitHub. 2019. Barista: An Event-centric Composable NOS for Software-Defined Networks (SDN Controller). Retrieved May 14, 2020 from https://github.com/sdx4u/barista.

[22] Djamila Bendouda, Abderrezak Rachedi, and Hafid Haffaf. 2018. Programmable architecture based on software defined network for Internet of Things: Connected dominated sets approach. *Future Generation Computer Systems* 80 (March 2018), 188–197.

[23] Samaresh Bera, Sudip Misra, and Athanasios V. Vasilakos. 2017. Software-defined networking for Internet of Things: A survey. *IEEE Internet of Things Journal* 4, 6 (Dec. 2017), 1994–2008.

[24] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, et al. 2014. ONOS. In *Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (HotSDN'14).* ACM, New York, NY, 1–6.

[25] Gary Berger. 2019. No.de Based OpenFLow Controller. Retrieved May 16, 2020 from https://github.com/gaberger/NodeFLow.

[26] Idris Z. Bholebawa and Upena D. Dalal. 2018. Performance analysis of SDN/OpenFlow controllers: POX versus floodlight. *Wireless Personal Communications* 98, 2 (2018), 1679–1699.

[27] Sujit Biswas, Kashif Sharif, Fan Li, Zohaib Latif, Salil S. Kanhere, and Saraju P. Mohanty. 2020. Interoperability and synchronization management of blockchain-based decentralized e-health systems. *IEEE Transactions on Engineering Management* 67, 4 (2020), 1–14.

[28] Sujit Biswas, Kashif Sharif, Fan Li, Sabita Maharjan, Saraju P. Mohanty, and Yu Wang. 2020. PoBT: A lightweight consensus algorithm for scalable IoT business blockchain. *IEEE Internet of Things Journal* 7, 3 (March 2020), 2343–2355.

[29] Sujit Biswas, Kashif Sharif, Fan Li, and Saraju P. Mohanty. 2020. Blockchain for E-health-care systems: Easier said than done. *Computer* 53, 7 (July 2020), 57–67.

[30] Sujit Biswas, Kashif Sharif, Fan Li, Boubakr Nour, and Yu Wang. 2019. A scalable blockchain framework for secure transactions in IoT. *IEEE Internet of Things Journal* 6, 3 (June 2019), 4650–4659.

[31] Andreas Blenk, Arsany Basta, Laurenz Henkel, Johannes Zerwas, Wolfgang Kellerer, and Stefan Schmid. 2018. perfbench: A tool for predictability analysis in multi-tenant software-defined networks. In *Proceedings of the ACM SIGCOMM Conference on Posters and Demos.* ACM, New York, NY, 66–68.

[32] Andreas Blenk, Arsany Basta, and Wolfgang Kellerer. 2015. HyperFlex: An SDN virtualization architecture with flexible hypervisor function allocation. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*. IEEE, Los Alamitos, CA, 397–405.

[33] Fabio Botelho, Alysson Bessani, Fernando M. V. Ramos, and Paulo Ferreira. 2014. On the design of practical fault-tolerant SDN controllers. In *Proceedings of the 3rd European Workshop on Software Defined Networks*. IEEE, Los Alamitos, CA.

[34] Mathieu Boussard, Dinh Thai Bui, Richard Douville, Pascal Justen, Nicolas Le Sauze, Pierre Peloso, Frederik Vandeputte, and Vincent Verdot. 2018. Future spaces: Reinventing the home network for better security and automation in the IoT era. *Sensors* 18, 9 (Sept. 2018), 2986.

[35] Mathieu Boussard, Serge Papillon, Pierre Peloso, Matteo Signorini, and Erez Waisbard. 2019. STewARD:SDN and blockchain-based Trust evaluation for automated risk management on IoT devices. In *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS'19)*. IEEE, Los Alamitos, CA, 841–846.

[36] Elif Bozkaya and Berk Canberk. 2020. SDN-enabled deployment and path planning of aerial base stations. *Computer Networks* 171 (April 2020), 107125.

[37] Zheng Cai, Alan L. Cox, and T. S. Eugene Ng. 2010. *Maestro: A System for Scalable OpenFlow Control*. Technical Report TR10-08. Rice University.

[38] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. 2007. Ethane: Taking control of the enterprise. *SIGCOMM Computer Communication Review* 37, 4 (Aug. 2007), 1–12.

[39] Martin Casado, Tal Garfinkel, Aditya Akella, Michael J. Freedman, Dan Boneh, Nick McKeown, and Scott Shenker. 2006. SANE: A protection architecture for enterprise networks. In *Proceedings of the USENIX Security Symposium*, Vol. 49. 50.

[40] Manisha Chahal and Sandeep Harit. 2019. Network selection and data dissemination in heterogeneous software-defined vehicular network. *Computer Networks* 161 (Oct. 2019), 32–44.

[41] Manisha Chahal, Sandeep Harit, Krishn K. Mishra, Arun Kumar Sangaiah, and Zhigao Zheng. 2017. A survey on software-defined networking in vehicular ad hoc networks: Challenges, applications and use cases. *Sustainable Cities and Society* 35 (Nov. 2017), 830–840.

[42] Subhrendu Chattopadhyay. 2019. Aloe. Retrieved May 14, 2020 from https://github.com/subhrendu1987/Aloe-Source.

[43] Subhrendu Chattopadhyay, Soumyajit Chatterjee, Sukumar Nandi, and Sandip Chakraborty. 2019. Aloe: An elastic auto-scaled and self-stabilized orchestration framework for IoT applications. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'19)*. IEEE, Los Alamitos, CA, 802–810.

[44] Djabir Abdeldjalil Chekired, Mohammed Amine Togou, Lyes Khoukhi, and Adlen Ksentini. 2019. 5G-slicing-enabled scalable SDN core network: Toward an ultra-low latency of autonomous driving service. *IEEE Journal on Selected Areas in Communications* 37, 8 (Aug. 2019), 1769–1782.

[45] Huang Ma Chi. 2019. PureSDN: Routing Application for FatTree Network. Retrieved May 12, 2020 from https://github.com/Huangmachi/PureSDN.

[46] China863SDN. 2019. An Open Source SDN Controller for Cloud Computing Data Centers. Retrieved May 16, 2020 from https://github.com/China863SDN/DCFabric.

[47] Shihabur Rahman Chowdhury, Md. Faizul Bari, Reaz Ahmed, and Raouf Boutaba. 2014. PayLess: A low cost network monitoring framework for software defined networks. In *Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS'14)*. IEEE, Los Alamitos, CA, 1–9.

[48] Christian Sieber. 2018. hvbench: An open and Scalable SDN Hypervisor Benchmark. Retrieved May 21, 2020 from https://github.com/tum-lkn/perfbench.

[49] Roberto Doriguzzi Corin, Matteo Gerola, Roberto Riggio, Francesco De Pellegrini, and Elio Salvadori. 2012. VeRTIGO: Network virtualization and beyond. In *Proceedings of the European Workshop on Software Defined Networking*. IEEE, Los Alamitos, CA, 24–29.

[50] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. 2011. DevoFlow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, New York, NY, 254–265.

[51] Mohamad Darianian, Carey Williamson, and Israat Haque. 2017. Experimental evaluation of two OpenFlow controllers. In *Proceedings of the IEEE 25th International Conference on Network Protocols (ICNP'17)*. IEEE, Los Alamitos, CA, 1–6.

[52] Tamal Das, Vignesh Sridharan, and Mohan Gurusamy. 2020. A survey on controller placement in SDN. *IEEE Communications Surveys & Tutorials* 22, 1 (2020), 472–503.

[53] Bruno Trevizan de Oliveira, Lucas Batista Gabriel, and Cintia Borges Margi. 2015. TinySDN: Enabling multiple controllers for software-defined wireless sensor networks. *IEEE Latin America Transactions* 13, 11 (Nov. 2015), 3690–3696.

[54] J. Van der Merwe, A. Cepleanu, K. D'Souza, B. Freeman, A. Greenberg, D. Knight, R. McMillan, et al. 2006. Dynamic connectivity management with an intelligent route service control point. In *Proceedings of the SIGCOMM Workshop on Internet Network Management (INM'06).* ACM, New York, NY, 29–34.

[55] Abdelouahid Derhab, Mohamed Guerroumi, Abdu Gumaei, Leandros Maglaras, Mohamed Amine Ferrag, Mithun Mukherjee, and Farrukh Aslam Khan. 2019. Blockchain and random subspace learning-based IDS for SDN-enabled industrial IoT security. *Sensors* 19, 14 (July 2019), 3119.

[56] Dmitry Zuikov, Alexander Vershilov, Kirill Zaborsky, Yury Shvedov, and Daria Zimarina. 2013. hcprobe: Framework for testing OpenFlow controllers. Retrieved May 22, 2020 from https://github.com/ARCCN/hcprobe.

[57] Docker Inc.2019. Docker Overview: Docker Documentation. Retrieved May 16, 2020 from https://docs.docker.com/engine/docker-overview/.

[58] Dmitry Drutskoy, Eric Keller, and Jennifer Rexford. 2013. Scalable network virtualization in software-defined networks. *IEEE Internet Computing* 17, 2 (March 2013), 20–27.

[59] Simon Duquennoy. 2019. Contiki OS. Retrieved May 12, 2020 from https://github.com/contiki-ng/contiki-ng/wiki.

[60] Tulio Dapper e Silva, Carlos F. Emygdio de Melo, Pedro Cumino, Denis Rosário, E. Cerqueira, and E. Pignaton de Freitas. 2019. STFANET: SDN-based topology management for flying ad hoc network. *IEEE Access* 7 (2019), 173499–173514. DOI: 10.1109/ACCESS.2019.2956724

[61] Mesut Gunes, Matthias Wahlisch, Thomas C. Schmidt, Emmanuel Baccelli, and Oliver Hahm. 2019. RIOT: The Friendly Operating System for the Internet of Things. Retrieved May 16, 2020 from https://www.riot-os.org/

[62] David Erickson. 2013. The Beacon OpenFlow Controller. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN'13).* ACM, New York, NY, 13–18.

[63] Joakim Eriksson. 2018. CoAP Documentation. Retrieved May 12, 2020 from https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-CoAP.

[64] ETSI. 2019. OSM Release 6. Retrieved May 12, 2020 from https://osm.etsi.org/wikipub/index.php/OSM_Release_SIX.

[65] Tungsten Fabric. 2019. Opencontrails Docs. Retrieved May 16, 2020 from https://github.com/tungstenfabric/opencontrails-docs.

[66] A. Farrel, J.-P. Vasseur, and J. Ash. 2006. *A Path Computation Element (PCE)-Based Architecture.* IETF RFC 4655. IETF.

[67] Daniel Farrell. 2014. OpenDaylight Performance. Retrieved Oct. 18, 2020 from https://slides.com/dfarrell07/opendaylight-performance/.

[68] Daniel Farrell. 2015. WCBench: CBench, Wrapped in Stuff That Makes It Useful. Retrieved May 12, 2020 from https://github.com/dfarrell07/wcbench.

[69] Ivan Farris, Tarik Taleb, Yacine Khettab, and Jaeseung Song. 2019. A survey on emerging SDN and NFV security mechanisms for IoT systems. *IEEE Communications Surveys & Tutorials* 21, 1 (2019), 812–837.

[70] Nick Feamster, Hari Balakrishnan, Jennifer Rexford, Aman Shaikh, and Jacobus Merwe. 2004. The case for separating routing from routers. In *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture.* 5–12.

[71] Andrew D. Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, and Shriram Krishnamurthi. 2013. Participatory networking: An API for application control of SDNs. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 327–338.

[72] Florian and Anika Schwind. 2016. OpenFlow Controller Analysis Tool. Retrieved May 16, 2020 from https://github.com/lsinfo3/ofcprobe.

[73] Laura Galluccio, Sebastiano Milardo, Giacomo Morabito, and Sergio Palazzo. 2015. SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'15).* IEEE, Los Alamitos, CA, 513–521.

[74] Jianbin Gao, Kwame Opuni-Boachie Obour Agyekum, Emmanuel Boateng Sifah, Kingsley Nketia Acheampong, Qi Xia, Xiaojiang Du, Mohsen Guizani, and Hu Xia. 2020. A blockchain-SDN-enabled Internet of vehicles environment for fog computing and 5G networks. *IEEE Internet of Things Journal* 7, 5 (May 2020), 4278–4291.

[75] Sahil Garg, Kuljeet Kaur, Georges Kaddoum, Syed Hassan Ahmed, and Dushantha Nalin K. Jayakody. 2019. SDN-based secure and privacy-preserving scheme for vehicular networks: A 5G perspective. *IEEE Transactions on Vehicular Technology* 68, 9 (Sept. 2019), 8421–8434.

[76] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. 2005. A clean slate 4D approach to network control and management. *ACM SIGCOMM Computer Communication Review* 35, 5 (2005), 41–54.

[77] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. 2008. NOX: Towards an operating system for networks. *ACM SIGCOMM Computer Communication Review* 38, 3 (2008), 105–110.

[78] Nikhil Handigol, Srinivasan Seetharaman, Mario Flajslik, Nick McKeown, and Ramesh Johari. 2009. Plug-n-Serve: Load-balancing web traffic using OpenFlow. In *Proceedings of ACM SIGCOMM Demos.* https://conferences.sigcomm.org/sigcomm/2009/demos/sigcomm-pd-2009-final26.pdf.

[79] Israat Tanzeena Haque and Nael Abu-Ghazaleh. 2016. Wireless software defined networking: A survey and taxonomy. *IEEE Communcations Surveys & Tutorials* 18, 4 (2016), 2713–2737.
[80] Hashicorp. 2020. Vagrant Github Repo.Retrieved May 22, 2020 from https://github.com/hashicorp/vagrant.
[81] Timothy Hinrichs, Natasha Gude, Martın Casado, John Mitchell, and Scott Shenker. 2008. *Expressing and Enforcing Flow-Based Network Security Policies.* Technical Report. University of Chicago.
[82] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, B. Naidu Kondapa, Chandan Bhagat, et al. 2018. B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication.*ACM, New York, NY, 74–87.
[83] Zakaria Abou El Houda, Abdelhakim Senhaji Hafid, and Lyes Khoukhi. 2019. Cochain-SC: An intra- and inter-domain DDoS mitigation scheme based on blockchain using SDN and smart contract. *IEEE Access* 7 (2019), 98893–98907.
[84] Victoria Huang, Qiang Fu, Gang Chen, Elliott Wen, and Jonathan Hart. 2017. BLAC: A bindingless architecture for distributed SDN controllers. In *Proceedings of the IEEE 42nd Conference on Local Computer Networks (LCN'17).* IEEE, Los Alamitos, CA, 146–154.
[85] Jonathan Hui, David Culler, and Samita Chakrabarti. 2010. *6LoWPAN: Incorporating IEEE 802.15. 4 into the IP Architecture.* White Paper. IEB Media.
[86] InMon Corp. 2019. sFlow-RT: Platform for Open Source Analytics Based Applications. Retrieved May 15, 2020 from https://sflow-rt.com/.
[87] Philippos Isaia and Lin Guan. 2016. Performance benchmarking of SDN experimental platforms. In *Proceedings of the IEEE NetSoft Conference and Workshops (NetSoft'16).* IEEE, Los Alamitos, CA, 116–120.
[88] Pedro Heleno Isolani, Juliano Araujo Wickboldt, Cristiano Bonato Both, Juergen Rochol, and Lisandro Zambenedetti Granville. 2015. SDN interactive manager: An OpenFlow-based SDN manager. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management.* IEEE, Los Alamitos, CA, 1157–1158.
[89] Wafa Ben Jaballah, Mauro Conti, and Chhagan Lal. 2020. Security and design requirements for software-defined VANETs. *Computer Networks* 169 (March 2020), 107099.
[90] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, et al. 2013. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 3–14.
[91] Michael Jarschel, Frank Lehrieder, Zsolt Magyari, and Rastin Pries. 2012. A flexible OpenFlow-controller benchmark. In *Proceedings of the European Workshop on Software Defined Networking.* IEEE, Los Alamitos, CA, 48–53.
[92] Michael Jarschel, Christopher Metter, Thomas Zinner, Steffen Gebert, and Phuoc Tran-Gia. 2014. OFCProbe: A platform-independent tool for OpenFlow controller analysis. In *Proceedings of the IEEE 5th International Conference on Communications and Electronics (ICCE'14).* IEEE, Los Alamitos, CA, 182–187.
[93] Younchan Jung, Marnel Peradilla, and Ronnel Agulto. 2019. Packet key-based end-to-end security management on a blockchain control plane. *Sensors* 19, 10 (May 2019), 2310.
[94] Ahmed Jawad Kadhim and Seyed Amin Hosseini Seno. 2019. Energy-efficient multicast routing protocol based on SDN and fog computing for vehicular networks. *Ad Hoc Networks* 84 (March 2019), 68–81.
[95] Hiroaki Kawai. 2018. Twink. Retrieved May 26, 2020 from https://github.com/hkwi/twink.
[96] Aliaksandr Kazarez. 2016. LOOM Controller. Retrieved May 16, 2020 from https://github.com/FlowForwarding/loom.
[97] Zuhran Khan Khattak, Muhammad Awais, and Adnan Iqbal. 2014. Performance evaluation of OpenDaylight SDN controller. In *Proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS'14).* IEEE, Los Alamitos, CA, 671–676.
[98] Rahamatullah Khondoker, Adel Zaalouk, Ronald Marx, and Kpatcha Bayarou. 2014. Feature-based comparison and selection of software defined networking (SDN) controllers. In *Proceedings of the World Congress on Computer Applications and Information Systems (WCCAIS'14).* IEEE, Los Alamitos, CA, 1–7.
[99] Woojoong Kim, Jian Li, James Won-Ki Hong, and Young-Joo Suh. 2016. OFMon: OpenFlow monitoring system in ONOS controllers. In *Proceedings of the IEEE NetSoft Conference and Workshops (NetSoft'16).* IEEE, Los Alamitos, CA, 397–402.
[100] Yeonkeun Kim, Jaehyun Nam, Taejune Park, Sandra Scott-Hayward, and Seungwon Shin. 2019. SODA: A software-defined security framework for IoT environments. *Computer Networks* 163 (Nov. 2019), 106889.
[101] Thomas Kohler, Frank Durr, and Kurt Rothermel. 2018. ZeroSDN: A highly flexible and modular architecture for full-range distribution of event-based network control. *IEEE Transactions on Network and Service Management* 15, 4 (Dec. 2018), 1207–1221.
[102] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, et al. 2010. Onix: A distributed control platform for large-scale production networks. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation.* 351–364.

[103] Diego Kreutz, Fernando M. V. Ramos, Paulo E. Verissimo, Christian E. Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2015. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE* 103, 1 (Jan. 2015), 14–76.

[104] Andre Kutzleb. 2015. ZSDN-Controller. Retrieved May 12, 2020 from https://github.com/zeroSDN/ZSDN-Controller/.

[105] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo. 2004. The SoftRouter architecture. *ACM HOT-NETS*. Retrieved September 29, 2020 from https://www.microsoft.com/en-us/research/publication/the-softrouter-architecture/.

[106] Zohaib Latif, Kashif Sharif, Maria K. Alvi, and Fan Li. 2018. Simulation standardization: Current state and cross-platform system for network simulators. In *Mobile Ad-Hoc and Sensor Networks*. Communications in Computer and Information Science, Vol. 747. Springer, 497–508.

[107] Zohaib Latif, Kashif Sharif, Fan Li, Md. Monjurul Karim, Sujit Biswas, and Yu Wang. 2020. A comprehensive survey of interface protocols for software defined networks. *Journal of Network and Computer Applications* 156 (April 2020), 102563.

[108] Byungjoon Lee, Sae Hyong Park, Jisoo Shin, and Sunhee Yang. 2014. IRIS: The Openflow-based recursive SDN controller. In *Proceedings of the 2014 16th International Conference on Advanced Communication Technology (ICACT'14)*. 1227–1231.

[109] He Li, Kaoru Ota, and Mianxiong Dong. 2019. LS-SDV: Virtual network management in large-scale software-defined IoT. *IEEE Journal on Selected Areas in Communication* 37, 8 (Aug. 2019), 1783–1793.

[110] Shengru Li, Daoyun Hu, Wenjian Fang, Shoujiang Ma, Cen Chen, Huibai Huang, and Zuqing Zhu. 2017. Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability. *IEEE Network* 31, 2 (March 2017), 58–66.

[111] Chu Liang, Ryota Kawashima, and Hiroshi Matsuo. 2014. Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers. In *Proceedings of the 2nd International Symposium on Computing and Networking*. IEEE, Los Alamitos, CA, 171–177.

[112] Ying-Dar Lin, Yu-Kuen Lai, Chen-You Wang, and Yuan-Cheng Lai. 2018. OFBench: Performance test suite on OpenFlow switches. *IEEE Systems Journal* 12, 3 (Sept. 2018), 2949–2959.

[113] Linux Foundation. 2018. OpenDaylight Project. Retrieved May 12, 2020 from https://wiki.opendaylight.org.

[114] Jose Ruiz-Mas, Jose Saldana, Julian Fernandez-Navajas, Jose Luis Almodovar, Luis Sequeira, and Juan Luis de la Cruz. 2018. Odin-Wi5. Retrieved May 12, 2020 from https://github.com/Wi5/odin-wi5/wiki.

[115] Guiyang Luo, Haibo Zhou, Nan Cheng, Quan Yuan, Jinglin Li, Fangchun Yang, and Xuemin Shen. 2020. Software defined cooperative data sharing in edge computing assisted 5G-VANET. *IEEE Transactions on Mobile Computing*. DOI: 10.1109/TMC.2019.2953163

[116] Stephen Mallon, Vincent Gramoli, and Guillaume Jourjon. 2016. Are today's SDN controllers ready for prime-time? In *Proceedings of the 41st IEEE Conference on Local Computer Networks (LCN'16)*. IEEE, Los Alamitos, CA, 325–332.

[117] Rahim Masoudi and Ali Ghaffari. 2016. Software defined networks: A survey. *Journal of Network and Computer Applications* 67 (May 2016), 1–25.

[118] Arturo Mayoral, Ricard Vilalta, Raul Muñoz, Ramon Casellas, and Ricardo Martínez. 2017. SDN orchestration architectures and their integration with Cloud Computing applications. *Optical Switching and Networking* 26 (Nov. 2017), 2–13.

[119] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (2008), 69–74.

[120] Sebastiano Milardo and Angelos-Christos Anadiotis. 2019. SDN-WISE Lab. Retrieved May 12, 2020 from https://github.com/sdnwiselab.

[121] Mininet Team. 2018. Mininet: An Instant Virtual Network on Your Laptop (or other PC) Mininet. Retrieved May 12, 2020 from http://mininet.org/.

[122] Sudip Misra and Niloy Saha. 2019. Detour: Dynamic task offloading in software-defined fog for IoT applications. *IEEE Journal on Selected Areas in Communications* 37, 5 (May 2019), 1159–1166.

[123] Matthew Monaco, Oliver Michel, and Eric Keller. 2013. Applying operating system principles to SDN controller design. In *Proceedings of the 12th ACM Workshop on Hot Topics in Networks (HotNets'13)*. ACM, New York, NY.

[124] Habib Mostafaei and Michael Menth. 2018. Software-defined wireless sensor networks: A survey. *Journal of Network and Computer Applications* 119 (Oct. 2018), 42–56.

[125] Jaehyun Nam. 2018. SDX4U: Software-Defined Everything for You. Retrieved December 14, 2019 from https://sdxdata.com/category/barista/

[126] Jaehyun Nam, Hyeonseong Jo, Yeonkeun Kim, Phillip Porras, Vinod Yegneswaran, and Seungwon Shin. 2018. Barista: An event-centric NOS composition framework for software-defined networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'18)*. IEEE, Los Alamitos, CA, 980–988.

[127] Jaehyun Nam, Hyeonseong Jo, Yeonkeun Kim, Phillip Porras, Vinod Yegneswaran, and Seungwon Shin. 2019. Operator-defined reconfigurable network OS for software-defined networks. *IEEE/ACM Transactions on Networking* 27, 3 (June 2019), 1206–1219.

[128] Musa Ndiaye, Adnan M. Abu-Mahfouz, and Gerhard P. Hancke. 2020. SDNMM—A generic SDN-based modular management system for wireless sensor networks. *IEEE Systems Journal* 14, 2 (June 2020), 2347–2357.

[129] Tri G. Nguyen, Trung Phan, Binh Nguyen, Chakchai So-In, Zubair Baig, and Surasak Sanguanpong. 2019. SeArch: A collaborative and intelligent NIDS architecture for SDN-based cloud IoT networks. *IEEE Access* 7 (2019), 7678–7694.

[130] Van-Giang Nguyen, Anna Brunstrom, Karl-Johan Grinnemo, and Javid Taheri. 2017. SDN/NFV-based mobile packet core network architectures: A survey. *IEEE Communications Surveys & Tutorials* 19, 3 (2017), 1567–1602.

[131] Anh Nguyen-Ngoc, Simon Raffeck, Stanislav Lange, Stefan Geissler, Thomas Zinner, and Phuoc Tran-Gia. 2018. Benchmarking the ONOS controller with OFCProbe. In *Proceedings of the 2018 IEEE 7th International Conference on Communications and Electronics (ICCE'18)*. IEEE, Los Alamitos, CA, 367–372.

[132] Jéferson Campos Nobre, Allan M. de Souza, Denis Rosário, Cristiano Both, Leandro A. Villas, Eduardo Cerqueira, Torsten Braun, and Mario Gerla. 2019. Vehicular software-defined networking and fog computing: Integration and design principles. *Ad Hoc Networks* 82 (Jan. 2019), 172–181.

[133] Northbound Networks. 2019. Zodiac FX OpenFlow Switch Hardware. Retrieved May 12, 2020 from https://github.com/NorthboundNetworks/ZodiacFX.

[134] NSNAM. 2019. NS-3, a Discrete-Event Network Simulator for Internet Systems. Retrieved May 12, 2020 from https://www.nsnam.org/wiki/Main_Page.

[135] NSNAM. 2019. OpenFlow Switch Support: Model Library. Retrieved May 12, 2020 from https://www.nsnam.org/docs/release/3.29/models/html/openflow-switch.html.

[136] Yustus Eko Oktian, SangGon Lee, HoonJae Lee, and JunHuy Lam. 2017. Distributed SDN controller system: A survey on design choice. *Computer Networks* 121 (July 2017), 100–111.

[137] Open Networking Foundation (ONF). 2019. Open Network Operating System (ONOS). Retrieved May 12, 2020 from https://onosproject.org/.

[138] Open Networking Foundation (ONF). 2012. OpenFlow Switch Specification, Version 1.3.0. Retrieved May 16, 2020 from https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf.

[139] OpenStack Team. 2019. OpenStack Havana. Retrieved May 16, 2020 from https://www.openstack.org/software/havana/.

[140] Haixia Peng, Qiang Ye, and Xuemin Sherman Shen. 2019. SDN-based resource management for autonomous vehicular networks: A multi-access edge computing approach. *IEEE Wireless Communications* 26, 4 (Aug. 2019), 156–162.

[141] Charles E. Perkins and Elizabeth M. Royer. 1999. Ad-hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*. IEEE, Los Alamitos, CA, 90–100.

[142] Kevin Phemius, Mathieu Bouet, and Jeremie Leguay. 2014. DISCO: Distributed SDN controllers in a multi-domain environment. In *Proceedings of the IEEE Network Operations and Management Symposium (NOMS'14)*. IEEE, Los Alamitos, CA.

[143] Mehran Pourvahab and Gholamhossein Ekbatanifard. 2019. An efficient forensics architecture in software-defined networking-IoT using blockchain technology. *IEEE Access* 7 (2019), 99573–99588.

[144] POX. 2018. POX Controller Manual. Retrieved May 12, 2020 from https://noxrepo.github.io/pox-doc/html/.

[145] Raspberry Pi Foundation. 2019. Raspberry Pi 3 Model B+. Retrieved May 12, 2020 from https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.

[146] Shailendra Rathore, Byung Wook Kwon, and Jong Hyuk Park. 2019. BlockSecIoTNet: Blockchain-based decentralized security architecture for IoT network. *Journal of Network and Computer Applications* 143 (Oct. 2019), 167–177.

[147] Danda B. Rawat. 2019. Fusion of software defined networking, edge computing, and blockchain technology for wireless network virtualization. *IEEE Communications Magazine* 57, 10 (Oct. 2019), 50–55.

[148] James Ray. 2019. Ethereum. Retrieved May 15, 2020 from https://github.com/ethereum/wiki/wiki.

[149] Yakov Rehkter, Tony Li, and Susan Hares. 2006. *A Border Gateway Protocol 4*. IETF RFC 4271. IETF.

[150] Wei Ren, Yan Sun, Hong Luo, and Mohsen Guizani. 2019. BLLC: A batch-level update mechanism with low cost for SDN-IoT networks. *IEEE Internet of Things Journal* 6, 1 (Feb. 2019), 1210–1222.

[151] Wei Ren, Yan Sun, Hong Luo, and Mohsen Guizani. 2019. A novel control plane optimization strategy for important nodes in SDN-IoT networks. *IEEE Internet of Things Journal* 6, 2 (April 2019), 3558–3571.

[152] RUNOS. 2019. RUNOS SDN and OpenFlow Controller. Retrieved May 16, 2020 from https://github.com/ARCCN/runos.

[153] Ryu SDN Framework Community. 2017. Ryu SDN Framework. Retrieved December 12, 2020 from https://github.com/faucetsdn/ryu.

[154] Ousmane Sadio, Ibrahima Ngom, and Claude Lishou. 2020. Design and prototyping of a software defined vehicular networking. *IEEE Transactions on Vehicular Technology* 69, 1 (Jan. 2020), 842–850.

[155] Dipjyoti Saikia, SeokHwan Kong, Nikhil Malik, and Dayoung Kim. 2015. OpenMuL SDN Platform. Retrieved May 16, 2020 from https://github.com/openmul/openmul.

[156] Ola Salman, Imad H. Elhajj, Ayman Kayssi, and Ali Chehab. 2016. SDN controllers: A comparative study. In *Proceedings of the 18th Mediterranean Electrotechnical Conference (MELECON'16)*. IEEE, Los Alamitos, CA, 1–6.

[157] Mohd Abuzar Sayeed, Rajesh Kumar, and Vishal Sharma. 2019. Efficient data management and control over WSNs using SDN-enabled aerial networks. *International Journal of Communication Systems* 33, 1 (Aug. 2019), e4170.

[158] Hichem Sedjelmaci and Sidi Mohammed Senouci. 2015. An accurate and efficient collaborative intrusion detection framework to secure vehicular networks. *Computers & Electrical Engineering* 43 (April 2015), 33–47.

[159] Syed Abdullah Shah, Jannet Faiz, Maham Farooq, Aamir Shafi, and Syed Akbar Mehdi. 2013. An architectural evaluation of SDN controllers. In *Proceedings of the IEEE International Conference on Communications (ICC'13)*. IEEE, Los Alamitos, CA, 3504–3508.

[160] Alexander Shalimov, D. Zuikov, D. Zimarina, Vasily Pashkov, and Ruslan Smeliansky. 2013. Advanced study of SDN/OpenFlow controllers. In *Proceedings of the 9th Central and Eastern European Software Engineering Conference.*ACM, New York, NY, 1–6.

[161] Ganesh H. Shankar. 2018. OFNet Quick User Guide. Retrieved May 16, 2020 from https://www.slideshare.net/monjurul88/ofnetsdn-controller-testing-framework.

[162] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. 2009. *FlowVisor: A Network Virtualization Layer.* Technical Report. OpenFlow Switch Consortium.

[163] Rob Sherwood and K.-K. Yap. 2011. Cbench Controller Benchmarker. Retrieved May 12, 2020 from https://github.com/andi-bigswitch/oflops/tree/master/cbench.

[164] Seungwon Shin, Yongjoo Song, Taekyung Lee, Sangho Lee, Jaewoong Chung, Phillip Porras, Vinod Yegneswaran, Jiseong Noh, and Brent Byunghoon Kang. 2014. Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, New York, NY, 78–89.

[165] Christian Sieber, Andreas Blenk, Arsany Basta, and Wolfgang Kellerer. 2016. hvbench: An open and scalable SDN network hypervisor benchmark. In *Proceedings of the IEEE NetSoft Conference and Workshops*. IEEE, Los Alamitos, CA, 403–406.

[166] Pranav Kumar Singh, Sahil Sharma, Sunit Kumar Nandi, and Sukumar Nandi. 2019. Multipath TCP for V2I communication in SDN controlled small cell deployment of smart city. *Vehicular Communications* 15 (Jan. 2019), 1–15.

[167] Christoph Sommer. 2019. Veins Vehicle in Network Simulations: The Open Source Vehicular Network Simulation Framework. Retrieved May 15, 2020 from http://veins.car2x.org/.

[168] Kalupahana L. K. Sudheera, Maode Ma, and Peter H. J. Chong. 2019. Link stability based optimized routing framework for software defined vehicular networks. *IEEE Transactions on Vehicular Technology* 68, 3 (March 2019), 2934–2945.

[169] Dongeun Suh, Seokwon Jang, Sol Han, Sangheon Pack, Myung-Sup Kim, Taehong Kim, and Chang-Gyu Lim. 2017. Toward highly available and scalable software defined networks for service providers. *IEEE Communications Magazine* 55, 4 (April 2017), 100–107.

[170] SUMO. 2019. SUMO User Documentation. Retrieved May 12, 2020 from https://sumo.dlr.de/docs/index.html.

[171] Yasuhito Takamiya and Nick Karanatsios. 2018. Trema OpenFlow Controller Framework. Retrieved May 12, 2020 from https://github.com/trema/.

[172] Xiaobo Tan, Hai Zhao, Guangjie Han, Wenbo Zhang, and Teng Zhu. 2019. QSDN-WISE: A new QoS-based routing protocol for software-defined wireless sensor networks. *IEEE Access* 7 (2019), 61070–61082.

[173] OpenStack Team. 2019. OpenStack. Retrieved May 16, 2020 from https://www.openstack.org/software/.

[174] Tryfon Theodorou and Lefteris Mamatas. 2017. CORAL-SDN: A software-defined networking solution for the Internet of Things. In *Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks*. IEEE, Los Alamitos, CA.

[175] Tryfon Theodorou, George Violettas, P. Valsamas, Sophia Petridou, and Lefteris Mamatas. 2019. A multi-protocol software-defined networking solution for the Internet of Things. *IEEE Communications Magazine* 57, 10 (Oct. 2019), 42–48.

[176] Benoit Thebaudeau. 2019. An Introduction to Cooja Simulator. Retrieved May 12, 2020 from https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja.

[177] Amin Tootoonchian. 2014. NOX Verity: The Last Version of the NOX Controller. Retrieved June 5, 2020 from https://github.com/noxrepo/nox.

[178] Amin Tootoonchian and Yashar Ganjali. 2010. HyperFlow: A distributed control plane for OpenFlow. In *Proceedings of the Internet Network Management Conference on Research on Enterprise Networking.*

[179] Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. 2010. OpenTM: Traffic matrix estimator for OpenFlow networks. In *Passive and Active Measurement*. Springer, 201–210.

[180] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, and Rob Sherwood. 2012. On controller performance in software-defined networks. In *Proceedings of the USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*.

[181] Truffle Blockchain Group. 2019. Sweet Tools for Smart Contracts: Truffle Suite. Retrieved May 15, 2020 from https://www.trufflesuite.com/.

[182] Hardeep Uppal and Dane Brandon. 2010. *OpenFlow Based Load Balancing*. Technical Report CSE561 Networking Project. University of Washington.

[183] Niels L. M. van Adrichem, Christian Doerr, and Fernando A. Kuipers. 2014. OpenNetMon: Network monitoring in OpenFlow software-defined networks. In *Proceedings of the IEEE Network Operations and Management Symposium (NOMS'14)*. IEEE, Los Alamitos, CA, 1–8.

[184] Vijay Varadharajan, Kallol K. Karmakar, and Udaya Tupakula. 2017. Securing communication in multiple autonomous system domains with SDN. In *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management*. IEEE, Los Alamitos, CA, 195–203.

[185] Bhuvaneswaran Vengainathan, Anton Basil, Mark Tassinari, Vishwas Manral, and Sarah Banks. 2018. *Benchmarking Methodology for Software-Defined Networking (SDN) Controller Performance*. IETF RFC 8456. IETF.

[186] Veryx Technologies. 2016. PktBlaster SDN Datasheet. Retrieved May 16, 2020 from https://www.veryxtech.com/wp-content/uploads/2015/10/Datasheet-PktBlaster-SDN-Controller-Test5.pdf.

[187] Allan Vidal, Christian Esteve Rothenberg, and Fábio Luciano Verdi. 2014. The libfluid OpenFlow driver implementation. In *Proceedings of the 32nd Brazilian Symposium on Computer Networks*. 1029–1036.

[188] George Violettas, Sophia Petridou, and Lefteris Mamatas. 2018. Routing under heterogeneity and mobility for the Internet of Things: A centralized control approach. In *Proceedings of the IEEE Global Communications Conference*. IEEE, Los Alamitos, CA, 1–7.

[189] Petra Vizarreta, Kishor Trivedi, Bjarne Helvik, Poul Heegaard, Andreas Blenk, Wolfgang Kellerer, and Carmen Mas Machuca. 2018. Assessing the maturity of SDN controllers with software reliability growth models. *IEEE Transactions on Network and Service Management* 15, 3 (Sept. 2018), 1090–1104.

[190] Andreas Voellmy and Junchang Wang. 2012. Scalable software defined network controllers. *ACM SIGCOMM Computer Communication Review* 42, 4 (Sept. 2012), 289–290.

[191] Anduo Wang, Xueyuan Mei, Jason Croft, Matthew Caesar, and Brighten Godfrey. 2016. Ravel: A database-defined network. In *Proceedings of the Symposium on SDN Research (SOSR'16)*. ACM, New York, NY.

[192] Junhua Wang, Kai Liu, Ke Xiao, Xiumin Wang, Qingwen Han, and Victor Chung Sing Lee. 2019. Delay-constrained routing via heterogeneous vehicular communications in software defined BusNet. *IEEE Transactions on Vehicular Technology* 68, 6 (June 2019), 5957–5970.

[193] Richard Wang, Dana Butnariu, and Jennifer Rexford. 2011. OpenFlow-based server load balancing gone wild. In *Proceedings of the USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*.

[194] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. 2015. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials* 17, 1 (2015), 27–51.

[195] Lixia Xie, Ying Ding, Hongyu Yang, and Xinmu Wang. 2019. Blockchain-based secure and trustworthy Internet of Things in SDN-enabled 5G-VANETs. *IEEE Access* 7 (2019), 56656–56666.

[196] Liu Xingtao, Guo Yantao, Wu Wei, Zhou Sanyou, and Li Jiliang. 2016. Network virtualization by using software-defined networking controller based Docker. In *Proceedings of the IEEE Information Technology, Networking, Electronic, and Automation Control Conference*. IEEE, Los Alamitos, CA, 1112–1115.

[197] Fangmin Xu, Huanyu Ye, Fan Yang, and Chenglin Zhao. 2019. Software defined mission-critical wireless sensor network: Architecture and edge offloading strategy. *IEEE Access* 7 (2019), 10383–10391.

[198] Lily Yang, Todd A. Anderson, Ram Gopal, and Ram Dantu. 2004. *Forwarding and Control Element Separation (ForCES) Framework*. IETF RFC 3746. IETF.

[199] Ibrar Yaqoob, Iftikhar Ahmad, Ejaz Ahmed, A. Gani, Muhammad Imran, and N. Guizani. 2017. Overcoming the key challenges to establishing vehicular communication: Is SDN the answer?*IEEE Communications Magazine* 55, 7 (2017), 128–134.

[200] Soheil Hassas Yeganeh and Yashar Ganjali. 2012. Kandoo: A framework for efficient and scalable offloading of control applications. In *Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks*. ACM, New York, NY, 19–24.

[201] Soheil Hassas Yeganeh and Yashar Ganjali. 2014. Beehive: Towards a simple abstraction for scalable software-defined networking. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks (HotNets'14)*. ACM, New York, NY, 1–7.

[202] Muhammad Usman Younus, Saif ul Islam, and Sung Won Kim. 2019. Proposition and real-time implementation of an energy-aware routing protocol for a software defined wireless sensor network. *Sensors* 19, 12 (June 2019), 2739.

[203] Alejandro Molina Zarca, Jorge Bernal Bernabe, Ruben Trapero, Diego Rivera, Jesus Villalobos, Antonio Skarmeta, Stefano Bianchi, Anastasios Zafeiropoulos, and Panagiotis Gouvas. 2019. Security management architecture for NFV/SDN-aware IoT systems. *IEEE Internet of Things Journal* 6, 5 (Oct. 2019), 8005–8020.

[204] Alejandro Molina Zarca, Dan Garcia-Carrillo, Jorge Bernal Bernabe, Jordi Ortiz, Rafael Marin-Perez, and Antonio Skarmeta. 2019. Enabling virtual AAA management in SDN-based IoT networks. *Sensors* 19, 2 (Jan. 2019), 295.

[205] Pan Zhang. 2016. MicroFlow: The Light-Weighted, Lightning Fast OpenFlow SDN Controller. Retrieved May 16, 2020 from https://github.com/PanZhangg/Microflow.

[206] Xiaoning Zhang, Shui Yu, Ji Zhang, and Zhichao Xu. 2019. Forwarding rule multiplexing for scalable SDN-based Internet of Things. *IEEE Internet of Things Journal* 6, 2 (April 2019), 3373–3385.

[207] Yuan Zhang, Lin Cui, Wei Wang, and Yuxiang Zhang. 2018. A survey on software defined networking with multiple controllers. *Journal of Network and Computer Applications* 103 (Feb. 2018), 101–118.

[208] Ning Zhao, Hao Wu, and Xiaonan Zhao. 2019. Consortium blockchain-based secure software defined vehicular network. *Mobile Networks and Applications* 25, 1 (June 2019), 314–327.

[209] Yimeng Zhao, Luigi Iannone, and Michel Riguidel. 2015. On the performance of SDN controllers: A reality check. In *Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN'15).* IEEE, Los Alamitos, CA, 79–85.

[210] Liehuang Zhu, Xiangyun Tang, Meng Shen, Xiaojiang Du, and Mohsen Guizani. 2018. Privacy-preserving DDoS attack detection using cross-domain traffic in software defined networks. *IEEE Journal on Selected Areas in Communications* 36, 3 (March 2018), 628–643.

[211] Zolertia. 2019. The Z1 Mote. Retrieved May 12, 2020 from https://github.com/Zolertia/Resources/wiki/The-Z1-mote.

Received December 2019; revised August 2020; accepted August 2020