Computer Networks 112 (2017) 279–293

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Review article

A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)

Murat Karakus*, Arjan Durresi

Department of Computer and Information Science, Indiana University Purdue University Indianapolis, Indianapolis, IN 46202, USA

ARTICLE INFO

Article history:
Received 6 May 2016
Revised 11 November 2016
Accepted 22 November 2016
Available online 23 November 2016

Keywords:
Scalability
Software-Defined Network
Control plane
Openflow
SDN
Survey

ABSTRACT

Software-Defined Networking (SDN) architecture has emerged in response to limitations of traditional networking architectures in satisfying today's complex networking needs. In particular, SDN allows network administrators to manage network services through abstraction of lower-level functionality. However, SDN is a logically centralized technology. Therefore, scalability, and especially the control plane (i.e. controller) scalability in SDN is one of the problems that needs more attention. In this survey paper, we first discuss the scalability problems of controller(s) in an SDN architecture. We then comprehensively survey and summarize the characterizations and taxonomy of state-of-the-art studies in SDN control plane scalability. We organize the discussion on control plane scalability into two broad approaches: Topology-related approaches and Mechanisms-related approaches. In Topology-related approaches, we study the relation between topology of architectures and scalability issues. It has sub-categories of Centralized (Single) Controller Designs and Distributed approaches. Distributed approaches, in turn, have also sub-categories: Distributed (Flat) Controller Designs, Hierarchical Controller Designs, and Hybrid Designs. In Mechanisms-related approaches, we review the relation between various mechanisms used to optimize controllers and scalability issues. It has sub-categories of Parallelism-based Optimization and Control Plane Routing Scheme-based Optimization. Furthermore, we outline the potential challenges and open problems that need to be addressed further for more scalable SDN control planes.

1. Introduction

Increasing cloud services, server virtualization, sharp growth of mobility and content-like video have led researchers to rethink today's network architectures. In traditional architectures, network devices and appliances are complex and difficult for (re)configuration and (re)installation since they require highly skilled personnel. Adding or moving a device from a network requires extra costs. It is also time-consuming because IT people need to deal with multiple switches, routers, etc. and update ACLs, VLANs and other mechanisms [1]. Furthermore, as business demands or user needs increase day by day, application developers, carriers, and enterprises delve into evolving new services and facilities. However, vendor dependency is an obstacle deterring them from developing new networking applications and services for their networks due to slow equipment product cycle, application testing and deployment. Therefore, today's data centers, carriers, and campuses need more dynamic architectures.

Software Defined Networking (SDN) [2–6] architecture has emerged in response to the aforementioned limitations of traditional networking architectures. SDN aims to decouple the control plane and data plane. This separation provides network operators/administrators with efficient use of network resources and eases provisioning of resources. Also, SDN brings ease of programmability in changing the characteristics of whole networks. This simplifies the management of the network, since it is decoupled from the data plane. Therefore, network operators can easily and quickly manage, configure, and optimize network resources with dynamic, automated and proprietary-free programs in SDN architecture [7]. Google's datacenter WAN, B4 [8], is one of the examples for SDN adopted in a large-scale network with the aforementioned purposes. In addition, since the network is logically centralized in SDN, controllers have a global visibility of the whole network unlike conventional networking. Hence, they can dynamically optimize flow-management and resources.

Despite the advantages of centralized control in SDN architectures, SDN faces some issues challenging its nature (i.e. centralized control) due to day by day increasing network demands. Although network operators enhance the performance of the network controllers, it still cannot be enough to meet the high network de-

* Corresponding author.
E-mail addresses: mkarakus@iupui.edu (M. Karakus), durresi@cs.iupui.edu (A. Durresi).

http://dx.doi.org/10.1016/j.comnet.2016.11.017
1389-1286/

mands such as flow request and monitoring network statistics. For example, one of the earlier SDN controllers, NOX [9], can serve only 30K flow requests per second with a response time less than 10 ms. This insufficiency appears more in large-scale networks or data centers compared to small networks. Kandula et al. [10] report that a cluster of 1500 servers receives 100K flows per second on average. Also, Erickson [11] states that a network with 100 switches can result in 10 million flow arrivals per second in the worst case. These numbers indicate that the control plane in an SDN architecture is prone to suffer from scalability issues due to its centralized nature. Furthermore, Sezer et al. [1] state that one of the main challenges in SDN is the scalability issue, which especially needs more attention by researchers. Therefore, understanding and improving the scalability of the SDN control plane (i.e. controller) is a critical problem for successful adoption of SDN for large scale networks or networks with many flows.

1.1. Survey organization

In this paper, we survey scalability problems of the control plane (i.e. controllers) in SDN architectures as opposed to other general SDN surveys. We discuss the main causes that make the control plane suffer from scalability issues in an SDN architecture. We also present characterizations and classifications of proposals based on the primary concepts exploited to alleviate the controller scalability issues. In addition, we point out the main challenges along with existing proposals in controller scalability.

We note that data plane scalability in SDN is not a part of this paper's scope. However, as a brief note, data plane scalability in SDN is mostly dominated by (1) processing power, (2) capacity of memory/buffer, and (3) software implementation of data plane devices. For a more detailed and comprehensive discussion on data plane scalability in SDN, we would like the readers to direct following studies: [12–19].

In the remaining sections of the paper, Section 2 gives a lightweight overview of the SDN framework with OpenFlow protocol. In Section 3, we discuss the *Scalability* concept regarding its meaning and present some scalability metrics proposed in the literature to quantitatively measure the scalability of systems both in general and SDN context as well as contributors to scalability issues in SDN. Section 4 presents our organization of the studies over control plane scalability in SDN. Section 5 outlines the relation between topology of architectures and scalability issues while Section 6 discusses the relation between other mechanisms used to optimize the controller performance and scalability issues. Section 7 presents a comparative discussion over control plane scalability proposals. In Section 8, we outline the potential challenges and open issues that need to be addressed further for fully scalable SDN control planes in the future in a nutshell. Finally, Section 9 wraps the paper up with concluding remarks.

2. An overview of SDN architecture and openflow protocol

SDN architecture with OpenFlow protocol enables network operators to treat flows in a finer-granular way compared to the traditional networks by means of controllers. In a traditional network, flows (or packets) are mainly treated based on a single or a few attribute combinations of packet headers, such as longest destination IP prefixes, destination MAC addresses, or a combination of IP addresses and TCP/UDP port numbers etc. SDN allows to manage flows based on more attributes of packet headers by means of a Controller-Data Plane Interface (C-DPI) such as OpenFlow protocol [20–23].

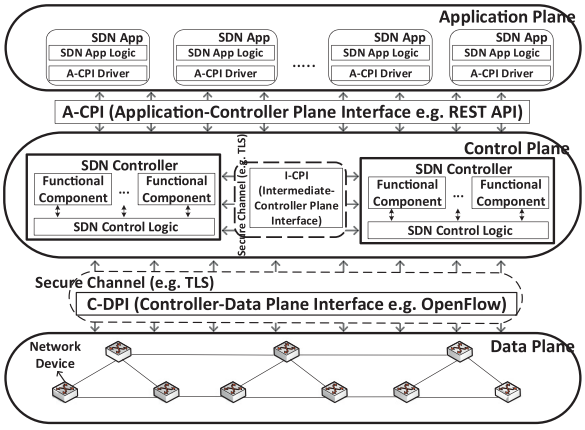


Fig. 1. An overview of an SDN control plane. Main components in a control plane of an SDN network are a controller(s) and interfaces (e.g. A-CPI, C-DPI, and I-CPI).

As shown in Fig. 1, Open Networking Foundation (ONF)[1] vertically splits SDN architecture into three main planes [24]:

- ***Data plane:*** The data plane is the bottom plane and consists of network devices such as routers, physical/virtual switches, access points etc. These devices are accessible and managed through C-DPIs by SDN controller(s). The network elements and controller(s) may communicate through secure connections such as the TLS connection. OpenFlow protocol is the most prevalent standard C-DPI used for communication between controller(s) and data plane devices.
- ***Control plane:*** An SDN control plane comprises a set of software-based SDN controller(s) to provide control functionality in order to supervise the network forwarding behavior through C-DPI. It has interfaces to enable communication among controllers in a control plane (Intermediate-Controller Plane Interface, i.e. I-CPI [25], optionally secured using the TLS), between controllers and network devices (C-DPI), and also between controllers and applications (Application-Controller Plane Interface, i.e. A-CPI). An A-CPI[2] renders possible the communication between network applications/services and controller(s) for network security, management etc. A controller consists of two main components: functional components and control logic. Controllers include more than one functional components such as Coordinator, Virtualizer etc. to manage controller behaviors. Furthermore, SDN control logic in a controller maps networking requirements of applications into instructions for network element resources [24].
- ***Application plane:*** An SDN application plane consists of one or more end-user applications (security, visualization etc.) that interact with controller(s) to utilize an abstract view of the network for their internal decision making process. These applications communicate with controller(s) via an open A-CPI (e.g. REST API). An SDN application comprises an SDN App Logic and A-CPI Driver.

In an SDN network with OpenFlow protocol and OpenFlow-enabled switches, there are three main parts in a switch: *Flow Table, Secure Channel*, and *OpenFlow Protocol*. An OpenFlow switch maintains a number of flow tables containing a list of flow entries. Each flow entry consists of 3 parts: A "Rule" field to define the flow entry based on certain header attributes such as source/destination addresses, an "Action" field to apply on a packet matching the values in the "Rule" field, and a "Stats" field to maintain some

[1] https://www.opennetworking.org/.

[2] An A-CPI is mostly called "Northbound Interface" by the SDN community.

counters for the entries [23]. A *Secure Channel* (e.g. TLS) is the interface that connects data plane elements to a remote controller. Switches are managed and configured by the controller over the secure channel. In addition, the controller receives events from the switches and sends packets out to switches through this channel.

In SDN, a controller can work in three operational modes to setup a new flow rule (a.k.a flow entry): reactive mode, proactive mode, and hybrid mode [26]:

- *Reactive mode*—In the reactive mode, when a new packet arrives to a network device (e.g. switch), the switch does a flow rule lookup in its flow tables. If no match for the flow is found, the switch forwards it to the controller using C-DPI so that the controller decides how to handle the packet. After the controller processes the packet according to the network policies, it creates and sends a flow entry to be installed in the network device. Future flows matching with this flow entry based on packet header attributes will be treated according to the corresponding matching rule.
- *Proactive mode*—In the proactive mode, flow entries are setup in flow tables of the switches before new flows arrive at the switches. When a packet arrives at a switch, the switch already knows how to deal with that packet. In this case, the controller is not involved in any flow rule setup process.
- *Hybrid mode*—In the hybrid mode, a controller benefits advantages of both reactive and proactive modes. It is quite possible that network administrators proactively install certain flow entries in data plane devices and the controller(s) reactively modify (delete/update) them or even add new flow entries based on incoming traffic.

While the proactive mode brings some concerns regarding inefficient use of switch memory, the reactive mode provides more agile, flexible, and dynamic environment for both controllers and switches [26].

2.1. Scalability support in openflow protocol

There are also some scalability related improvements in OpenFlow specifications. One improvement is the *group table*[3] *mechanism* specified in version 1.1 [27] and later. This mechanism enables multiple flow table entries to point to the same group identifier, so that the group table entry is performed for multiple flows. For example, if you need to update the action on this set of flow table entries (all have the same action), the controller can only update the pointed group table entry action instead of updating the action of all flow table entries. Another improvement is that it provides multiple controller support as of its version 1.2 [28] through the *controller role change mechanism*. This scheme enables a switch to establish communication with a single controller or multiple controllers in parallel under different controller roles such as master, equal, and slave.

3. Scalability and its causes in SDN

Scalability is a frequently-claimed attribute of various systems. It is a multi-dimensional topic. While the basic notion is intuitive, the term *scalability* does not evoke the same concept to everybody. Therefore, there is no general precise agreement on neither its definition nor content. While some people may refer to scalability as optimization of processing power to CPUs, others may define it as a measure of parallelization of applications across different machines. However, regardless of its meaning to someone, it is a desired property indicating positive sense regarding a system, architecture, algorithm and so on.

Furthermore, trade-offs concerning some concepts such as performance, resiliency, availability, reliability and flexibility have to be taken into account by network designers and managers while designing a network architecture [29]. A "solution" proposed as a scalability cure for a network may introduce trade-offs that harm other useful properties of the network. For example, in the context of SDN, proactive rule installation in SDN switches decreases the load of the controller, thus reducing the processing time and flow initiation overhead in the controller. However, this constraints the flexibility coming from reactive flow installation and reduces decision-making dynamicity of the controller and management of the network. Also, controller distribution is one way to overcome computational load on controller but it brings consistency and synchronization problems as well. Therefore, scalability is not an independent problem that can be exclusively dealt with but is a combination of issues that introduces trade-offs to be explicitly stated while proposing a remedy.

3.1. Existing scalability metrics in general

There are several research efforts [30–36] proposing a metric to measure scalability of systems. Most of these metrics are for homogeneous environments. The majority of these proposals revolve around two major types of scalability metrics: *Isospeed* scalability and *Isoefficiency* scalability.

The *Isospeed* scalability is characterized by the fact that an achieved average unit speed of an algorithm on a given machine can remain constant with increasing number of processors and problem size for an algorithm-machine combination [30]. In [31], the authors present a metric to describe the scalability of an algorithm-machine combination in homogeneous environments. Their scalability function is defined as $\psi(p, p') = \frac{p'W}{pW'}$ where p and p' are the initial and scaled number of processors of the systems respectively, and W and W' are the initial and scaled problem size (workload) respectively.

The *Isoefficiency* scalability is described as the ability of parallel machine to keep the parallel efficiency constant when the system and problem size increase [32]. The parallel efficiency is defined as speedup over the number of processors, i.e. $E = \frac{S}{p}$. Speedup is also given by the ratio of problem size (W) and parallel execution time (T_p), i.e. $S = \frac{W}{T_p}$ where $T_p = \frac{W+T_0(W,p)}{p}$ with $T_0(W, p)$ extra communication overhead [33].

Pastor and Orero [34] define heterogeneous scalability by presenting a heterogeneous efficiency function. They attempt to extend the homogeneous *Isoefficiency* scalability model to heterogeneous computing and, therefore, their work inherits the limitation of parallel speedup, requiring the measurement of solving large-scale problem on single node. Sun et al. [35] propose a scalability metric called *Isospeed-efficiency* for general heterogeneous computing systems. This metric combines the roots of both *Isospeed* scalability and *Isoefficiency* scalability metrics by means of a concept called "Marked Speed" to describe the computing power for a stand-alone node and a combined computing system.

3.2. Scalability in SDN

In SDN networks, controller performance is one of the primary concerns while designing more scalable networks. There are many studies exploring performances of controllers with respect to different network workload, implementations, architectures and so on [37–41]. Although studies evaluate scalability performance of controllers they propose regarding various performance metrics, such as path installation time, link utilization, and so on, depending on

[3] A group table consists of group entries. A group entry consists of a group identifier, a group type, counters, and a list of action buckets.

their target problem, the most prominent and considered metrics are control plane *throughput*, which refers to the number of flow requests handled per second, and (flow setup) *latency*, which refers to the delay to respond flow requests, in SDN context. In SDN, a controller needs to proactively or reactively set up (i.e. handle) and tear down flow-level forwarding state in OpenFlow switches. Once set up, the flow forwarding state remains cached on the OpenFlow switches so that this process is not repeated for subsequent packets in the same flow. This setup process includes a latency as well. It is perceived that this flow setup process is to be likeliest source of control plane (i.e. controller) performance bottleneck by the SDN community. Hence, the number of flow requests handled per second (throughput) and flow setup latency come into prominence in evaluation of control plane scalability performance. Therefore, the term *Scalability*, particularly control plane scalability in SDN context, is characterized by the aforementioned two metrics, throughput and flow setup latency, as well as in this paper. A more detailed comparison of studies in terms of their scalability performance in terms of throughput and flow setup latency metrics is given in Section 7.

There are also few research efforts proposing a metric to quantify scalability of SDN networks. Hu et al. [42] present a metric for SDN control plane scalability. They use the scalability metric, which is based on productivity of a distributed system, presented in [36] to quantify the scalability of SDN control plane by adapting to the SDN case. A similar work in [43] also proposes a metric to quantify control plane scalability by ratio of workload (number of flows entering the network through the data plane) and overhead (number of messages processed in the control plane). However, we should note that these metrics have been proposed recently. Therefore, we will see their adoption by SDN research community as one of the metrics for scalability performance measurement by the time.

3.3. Contributors to scalability issues in SDN

SDN is a logically centralized architecture, therefore scalability is one of the crucial issues to be addressed in SDN as in many traditional networks [44]. However, in particular, scalability concerns of the control plane in SDN are intrinsic to SDN owing to its separated structure. In this section, we point out the main reasons that make the control plane a scalability bottleneck in SDN.

- ***Separation of control plane and data plane:*** The separation of the data plane and control plane is a contributor to scalability issues of the SDN architecture, particularly control plane scalability, since this decoupling requires the management of network devices from a remote controlling mechanism (i.e. controller). Since data plane devices have no longer ability to make decisions about traffic packets a communication has to be established with controllers to receive corresponding decisions about the packets. This communication brings extra message burden for both controllers and data plane devices. Therefore, this separation may result in significant signaling overhead between control plane and data plane, depending on the network architecture (e.g. distributed, hierarchical etc.) and applications on top of the controller. Hence, this makes the control plane play a bottleneck role regarding the scalability of the system.
- ***Quantity of events/requests handled by a controller:*** This problem pertains more to the single controller designs than to the distributed (flat), hierarchical or hybrid designs since it results from the centralization of computation at a single central entity. An increase in the number of network devices reinforces the foregoing problem for controllers. As the network grows with respect to the size of the nodes (e.g. hosts, switches etc.), the controller will have to cope with more events and flow requests, which can make the controllers a bottleneck point due to its limited computation resources such as CPU and memory. Therefore, the number of control messages sent by data plane devices to the controller(s) becomes one point to be addressed because the controller may not be able to handle all the incoming requests. For example, a NOX controller can handle up to 30K requests/sec, which is enough for small to mid-size networks [45]. However, that number may not be enough for some network settings, such as data centers, depending on the number of servers and the switches [10,46]. This issue may also result in delay in programming of data-plane (devices) since it may increase flow rule setup process delay at controller, which eventually affects the speed of the network.
- ***Controller-switch communication delay:*** As stated in [47], the controller's placement (distance between network devices and controller) is one of the factors that introduces latency into the flow setup time. Flow setup latency is typically determined by switch packet processing time, RTT (*round-trip-time*) between controller and switches, and controller packet processing time. If the controller-switch communication delay (determined by RTT) is high, then resulting flow setup latency becomes high too, which causes longer flow rule addition, deletion or update in switch flow tables. This, in turn, may result in congestion in both control plane level and data plane level and longer failover time in the network. Hence, scalability of the controller degrades. Although this delay depends on physical distance between controllers and data plane devices, a well-defined placement of controllers may minimize the delay. This particularly becomes important in WAN compared to small scale networks. Azodolmolky et al. [48] outline a comprehensive analytical model for the behavior of a scalable SDN deployment regarding boundary performance of event processing delay and buffer space of SDN controllers by means of the network calculus as a mathematical framework.

SDN brings the possibility of various network innovations, but lacks uniform definitions and standard implantation in reality. Many essential issues of the controller (plane), however, need to be well addressed so as to improve the development and usages of SDN.

4. Classification of control plane scalability proposals

As discussed in Section 3, there is no consensus on the definition of scalability. Therefore, it is not easy to present an unified classification for scalability solutions. The organization that we present in this paper reflects our own point of view over the proposed studies in SDN control plane scalability.

As shown in Fig. 2, we organize the discussion on control plane scalability into two broad approaches. The first approach is Topology-related Approaches with sub-categories of Centralized (Single) Controller Designs and Distributed approaches. In this category, we study the relation between topology of architectures and scalability issues. Distributed approaches are Distributed (Flat) Controller Designs, Hierarchical Controller Designs and Hybrid Designs. Reducing the workload on a controller will result in a better performance of the controller regarding scalability. Therefore, distribution of control plane (i.e. controller) workload among controllers is one way related to the scalability. Hybrid designs represent the studies that leverage the data plane by devolving some limited control functions to the switches to partition the control plane workload. This approach is hybrid due to involvement of both the control plane and data plane in the network control. It differs from the distributed (flat) and hierarchical designs in the way that switches are involved in decision processing and network

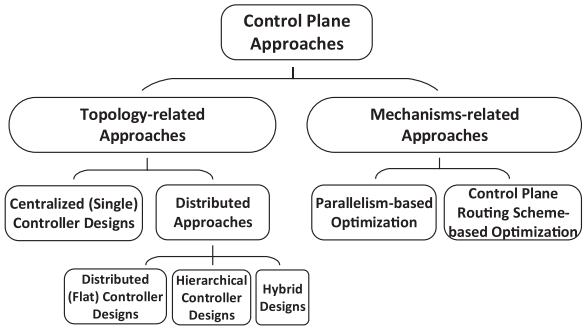


Fig. 2. Taxonomy of control plane approaches in SDN. The proposed approaches are categorized into two categories with sub-categories. Topology-related approaches revolve around structure of the framework to distribute the total workload that the controllers handle. Mechanisms-related approaches offer different ways of optimization for controllers and application implementations.

Table 1
Network types targeted by the studies. Most of the proposals target data centers (DC), enterprise networks or WAN networks.

Proposals \ network types	Campus	Cloud	DC	Enter prise	WAN
Beacon [11]				✓	
DEFO [49]					✓
DevoFlow [50]			✓		
DIFANE [51]				✓	
DISCO [52]			✓	✓	✓
D-SDN [53]					✓
ElastiCon [54]			✓	✓	
Ethane [55]	✓			✓	
Fibbing [56]	✓	✓	✓	✓	✓
FlowBroker [57]					✓
HyperFlow [58]		✓	✓		
Kandoo [59]	✓	✓	✓		
Logical xBar [60]					✓
Maestro [61]			✓		✓
McNettle [62]				✓	
NOX [9]			✓	✓	
NOX-MT [37]			✓	✓	
Onix [63]		✓	✓	✓	✓
ONOS [64]				✓	✓
Orion [65]					✓
Tavakoli et al. [66]			✓		
Tam et al. [67]			✓		
Yazici et al. [68]	✓		✓		
Bari et al. [69]					✓
Karakus et al. [70]					✓
Owens et al. [71]	✓	✓		✓	✓
Soliman et al. [72]					✓

control. We explain these approaches further in the corresponding subsections throughout the paper.

In the second approach, Mechanisms-related Approaches, we review the relation between various mechanisms used to optimize controllers and scalability issues. Enhancing controllers with respect to their performance by some optimization techniques results in better scalability performance too. In addition, reducing the events resulting from routing mechanism of a controller is another way to increase the scalability in control plane since routing process brings worth considering load to controller.

Also, some proposals may seem to belong more than one category. Hence, we classify and present such proposals by mainly focusing on their primary approaches.

Table 1 shows the network types targeted by the studies. Most of the proposals target data centers, enterprise networks or WAN networks since these networks are more vulnerable to the control plane scalability issues.

5. Topology-related approaches

In this approach, we review the relation between topology of architectures and scalability issues. The proposals that use different topology models, illustrated in Fig. 3, can be classified in four prevalent architectures: Centralized (Single) Controller Designs, Distributed (Flat) Controller Designs, Hierarchical Controller Designs, and Hybrid Designs. These designs have their own intrinsic advantages and disadvantages with respect to control plane scalability. We explain these architectures and present the related studies in corresponding subsections below.

5.1. Centralized (single) controller designs

This type of architecture settings revolve around a single central controller [9,55] with a global network view. The design of this architecture is simple and it is easy to manage the network. This design may meet the needs of small to mid-size networks. However, it is not efficient to handle the burden of environments such as data centers and large-scale networks due to number of events/requests that the controller must handle as stated in the Section 3. Therefore, a single controller design is considered less scalable compared to distributed (flat) controller, hierarchical controller and/or hybrid designs.

The authors in [55] develop a new networking architecture called “Ethane” that targets the enterprise networks although it is first deployed in campus network. In an Ethane network, network managers are able to define policies and each request that is not matching a flow entry has to traverse through the controller. There are three concerns that the authors address and resolve in this architecture. First, Ethane renders that high-level policies become the authority part to control the network. Second, the packet paths are managed by policies in order to have better control and global network view. Third, the Ethane network requires a precise binding between a packet and its origin to be able to identify where the packet coming is from.

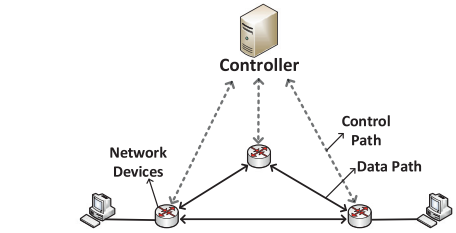
NOX [9] is inspired by the need for a centralized and uniform programmatic interface that would make a network more manageable. NOX is a network operating system that is more than just a controller platform for a network. As in most SDN controller platforms, NOX treats the packets based on the first packet of a flow traversing through the controller. This flow-based method helps in having more granular control over the traffic in a network. In [66], the authors investigate whether generalized solutions such as NOX can handle characteristic requirements of specialized environments such as datacenters.

5.2. Distributed approaches

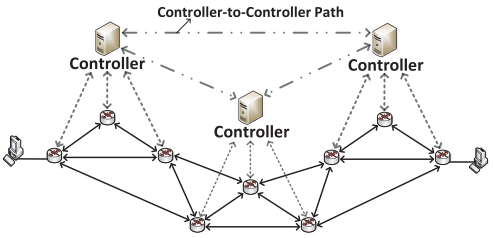
In this approach, we classify and present the studies [51–56] that distribute the control plane workload on controllers based on topological models, such as flat, hierarchical as well as hybrid designs. As using distributed controllers brings advantages such as load distribution and avoiding centralized (single) controller failure, it brings some challenges such as overhead from controller communication, latency due to state synchronization, and (policy/state) consistency among controller instances that are being addressed by researchers. We discuss these challenges in Section 8.

5.2.1. Distributed (flat) controller designs

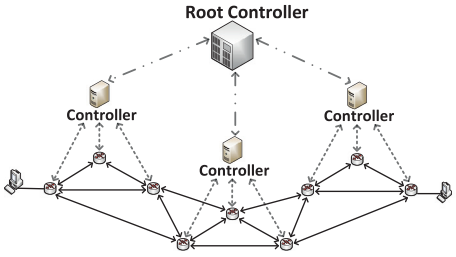
In this structure, each controller manages a sub-network/domain of the whole network. There are two strategies for distributed controller architectures to implement controller's network view. In the *local view strategy*, each controller has its



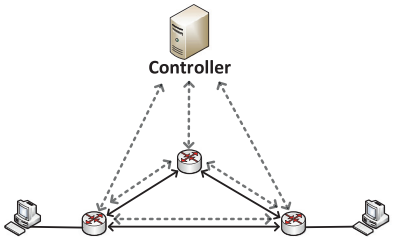
(a) Centralized (Single) Controller Design



(b) Distributed (Flat) Controller Design



(c) Hierarchical Controller Design



(d) Hybrid Design

Fig. 3. An overview of topology-related architectures. The two-sided solid, dashed, and dashed-dotted arrows represent two-way data path among network devices, control path between controller and data devices, and controller-to-controller path among controllers, respectively. In 3a (Centralized (Single) Controller Design), there is one main controller with global network state. In 3b (Distributed (Flat) Controller Design), every controller is responsible for different sites/parts of network(s) with partial or full shared network view. In 3c (Hierarchical Controller Design), there are levels in which controllers are responsible for different sites (sub-domains) and a Root controller on top with global network view for global applications like routing. In 3d (Hybrid Design), data plane devices are also involved in network control.

own local network view and each of its neighboring local networks is abstracted as a logical node. In the *global view strategy*, on the other hand, each controller has a global view of the whole network. In both cases, the controllers need to communicate through controller-to-controller channels to exchange needed state information (e.g. reachability information) regarding their domains.

HyperFlow [58] is logically centralized albeit its distributed architecture is an event-based control plane for OpenFlow. In HyperFlow, the authors exploit local controllers, serving all requests for their own remote sites, due to an increase in the flow setup times and flow initiation rates. It is actually implemented as a NOX [9] application that is responsible for: (a) global network view synchronization between controllers, (b) communication to switches controlled by another controller from a different site, and (c) managing responses coming from switches in other sites to the request-originator controllers. A system called "publish/subscribe" message paradigm is exploited to accomplish these tasks through controllers from different sites.

In [63], the authors propose a new distributed network platform called "Onix" for large-scale networks in response to deficiencies (e.g. providing consistent network state distribution, global network view among network applications, and failure recovery mechanisms) in a common control platform. Onix instances propagate network states to other instances to be able to scale large networks. The authors follow three approaches to improve scalability in Onix architecture; (1) Network Information Base (NIB)[4] partitioning by controller instances for less work, (2) cluster aggregation for a hierarchical structure, and (3) consistency and durability of the network states for applications. A similar work, Software Transactional Networking (STN) [73], also proposes a distributed control plane along with a scheme with a middleware to resolve policy consistency among distributed controller over the data plane. While Onix expects application writers to provide the necessary logic to detect and resolve conflicts of network state due to concurrent control, STN propose concurrent policy composition mechanisms that can be used by any application in a general fashion.

Tam et al. [67] study the feasibility of using multiple controllers to improve scalability without global network view and limited network topology information stored in controllers in a data center environment. They leverage flow routing example to see practicability of these controllers and propose two approaches, *path-partition* and *partition-path*, for the corresponding purpose.

In [68], the authors propose a distributed cluster-based controller architecture and a framework to retain the communication and coordination between controllers to obtain a more scalable network. This cluster-based architecture brings flexibility to the network regarding adding or removing controllers since it does not involve network applications. The controllers select a master controller that is in charge of delineation between controllers and switches.

Distributed controller architectures are proposed to mitigate the scalability issues of SDN networks. However, distributed controller architecture may not achieve the planned scalability because of the unbalanced load across the controllers since network administrators decide which and how many switches connect to a controller when they setup the network. Therefore, this may cause an overload in the controller.

ElastiCon [54] distributes the workload evenly through the controllers by means of a controller pool. This elastic distributed controller architecture dynamically shifts the workload across the controllers by adding or removing controllers to the controller pool and/or rebalancing the load of an individual controller based on threshold values.

Phemius et al. [52] present the "DISCO" (DIstributed Sdn COntrol plane) framework consisting of multiple controllers controlling different SDN domains that share aggregated network-wide information for a consistent network view on each controller. The

[4] NIB is a data structure to store network state and is roughly analogous to the Routing Information Base (RIB) used by IP routers.

DISCO framework has two main parts. While the intra-domain part is responsible for controller's own domain functionalities, the inter-domain part manages the flows across the distributed networks by exchanging the aggregated network state information such as reservation, topology etc. The difference of the DISCO framework from the other distributed architectures is its capability of differentiation of intra-domain and inter-domain information along with heterogeneous inter-domain links such as MPLS tunnels and SATCOM links.

Bari et al. [69] address difficulties of deploying multiple distributed controllers in a large-scale WAN network. They present a framework that readjusts the required active controllers with some assigned switches in accordance with current network dynamics to reduce flow setup time, horizontal overhead (between controllers) and vertical overhead (between controllers and switches). Their proposed management framework is responsible for (re)assignment of switches to controllers in case of a need.

ONOS [64] is another distributed SDN control platform aimed at improving scalability, performance and availability of networks. ONOS addresses how a network OS can scale horizontally to avoid becoming a performance bottleneck and avoid being a single point of failure. In ONOS, a large-scale WAN network can be divided into multiple parts controlled by different ONOS instances. These distributed ONOS instances construct a global network view for the network.

It is worth to mention another collaborative, open-source controller platform, the "OpenDaylight" (ODL) project [74]. The ODL is a Linux Foundation collaborative project to promote use of SDN. The ODL community has come together to establish an open reference controller framework to freely program and control an SDN architecture.

5.2.2. Hierarchical controller designs

In hierarchical architectures [53,57,59,60,65,70] local controllers handle local applications' requirements with frequent events, and a main more powerful controller, usually called as "Root", deals with non-local applications' needs requiring global network view and rare events as opposed to local controllers. Although controllers may have a global view of the whole network in the distributed (flat) controller designs, lower-tier controllers (which are more localized compared to upper-tier controllers) do not maintain a global view of the network in the hierarchical controller designs. Therefore, this design is different from the distributed (flat) design regarding network views of the controllers.

Kandoo [59] focuses on scaling a controller by decreasing the number of frequent events on the control plane since these events bring more overhead than others to the control plane. Kandoo's architecture comprises of two layers to sustain scalability. The bottom layer consists of local controllers which are not connected to each other and do not maintain a network wide state while the top layer is a logically centralized controller, connected to all bottom layer controllers, with the global network view. Frequent and resource-greedy events like flow arrivals are processed by the local controllers at the bottom layer, thereby preventing the root (top layer) controller from coping with more numbers of events.

McCauley et al. [60] discuss "Logical xBar" that is a recursive building block used to construct a centralized abstract hierarchical control plane. It exploits the idea of aggregating smaller units for forwarding into larger ones. The proposed control plane design has two building blocks: 1) Logical xBar, which is a programmable entity that can switch packets between ports, and 2) Logical Server which handles the forwardingtable management and the control plane computations. In the proposed design, the network itself does not necessarily need to be physically hierarchical, instead aggregation of logical xBars and logical Servers bring that abstracted hierarchy on the network.

Flat and hierarchical control plane structures may still suffer from certain issues. In flat control plane architecture, the controllers may face increasing computational complexity resulting from growing large size networks. On the other hand, the centralized hierarchical architectures suffer from path stretch problems [75].

In [65], the authors propose the "Orion", a hierarchical control plane for large-scale networks managed by the same administrator to alleviate the above-mentioned two problems. Orion has three layers: the bottom layer consists of network devices of areas; the middle layer consists of area controllers; and the top layer contains sub-domain controllers. Sub-domain controllers have global network views for their own domains and synchronize this information with each other by a distributed protocol.

In [53], the authors introduce Decentralize-SDN, D-SDN, framework that distributes a control plane not only physically but also logically in a SDN. D-SDN exploits the hierarchy of controllers in which main controllers (upper layer) delegate control to secondary controllers (bottom layer) to manage certain network devices.

Marconett and Yoo [57] propose the "FlowBroker" architecture for a better collaboration between multiple domains in terms of load balancing and network performance. The FlowBroker architecture exploits the idea of hierarchy with domain controllers and one or more super-controllers, called as Brokers, atop. Each domain controller may attach to more than one Broker according to their reputations that reflect performance of a Broker regarding load balancing and reliability. The FlowBroker architecture allows Brokers to cooperate between them to share abstracted network states coming from the domain controllers below level. They report that as the domain count increases from 6 to 10, the difference between utilizing 1 broker or 5 broker agents equals a 5–8% decrease in maximum link utilization, a 28–84% reduction in end-to-end delay, and 69–151% reduction in traffic loss.

Karakus and Durresi [70] propose a hierarchy-based network architecture along with an inter-AS routing approach with QoS. The authors use an idea of levels in which networks with controllers reside on top. There is also a main controller that works like a broker on top of networks to keep the global network state and view. Their experiment results indicate that a network controller in a hierarchic setting handles 50% less number of traffic than a network controller in a non-hierarchic environment.

5.2.3. Hybrid approach

This approach differs from the distributed (flat) and hierarchical designs in a way that data plane devices are involved in decision processing and network control. Therefore, this approach is considered hybrid due to involvement of both the control plane and data plane in the network control. In this subsection, we present several SDN architectures [50,51,56] that leverage the data plane by devolving some limited control functions (such as sending rules to other network devices to be added, deleted or updated in their flow tables etc.) to network devices for control plane workload partitioning, thereby improving scalability. This might happen either by installing rules proactively or reactively in the switches. Also, it is obvious that keeping flows as much as possible in the data plane reduces overhead and improves the controller performance regarding throughput and latency.

"DIFANE (DIstributed Flow Architecture for Networked Enterprises)" [51] is an architecture that preserves traffic in the data plane through managing packets in switches called "Authority Switches". In DIFANE, the authority switches are assigned rules by means of the controller that maintains an algorithm to partition the rules and minimizes rule fragmentation along with the authority switches.

"DevoFlow (Devolved Flow)" [50] addresses frequent interactions between the control plane and the data plane for the sake

of full control and global view over the network. Since this redundant interaction on almost every flow setup brings extra overhead and delay, the authors propose DevoFlow to reduce the interaction while preserving the required amount of visibility by conveying some functionalities of the control plane to the data plane. More efficiency and scalability are achieved because the controller controls only significant, and long-lived flows such as elephant flows. Use of wild-card rules which aggregate multiple rules into one minimizes the controller-switch communication as well. DevoFlow lets the switches make local decisions through cloning rules, multi-path support, and re-routing. However, there are some issues that remain open in DevoFlow such as how to manage some network applications including QoS, security, and traffic engineering.

Fibbing [56,76] is another hybrid SDN architecture that applies a central control over traditional distributed link-state protocols such as OSPF and IS-IS. In Fibbing architecture, the controller is still centralized and responsible for path computation based on requirements from operators as in SDN case. However, the actual computation of Forwarding Base Information (FIB) entries and their installation on data plane devices is done by the distributed control plane of traditional protocols run on the network. In this way, Fibbing takes advantages of centralized control (SDN) and distributed traditional protocols for scalability.

6. Mechanisms-related approaches

In Section 5, we have discussed topology-related approaches. In this section, we discuss other mechanisms used to optimize controller(s). Mechanisms-related approaches primarily exploit various optimization techniques in order to alleviate the foregoing scalability issues in SDN networks. They aim to empower the controller performance so that it can handle more packet flows per second (i.e. throughput), improve the latency, and reduce overhead. One way to increase throughput and improve latency is to exploit the parallelism in multi-core systems by means of some methods such as multi-threading, I/O batching etc. while another way is reducing the events processed in the control plane. These events mostly result from routing decisions made by the controller(s). Some research efforts propose better optimized routing decision mechanisms to reduce events to be processed in the control plane.

6.1. Parallelism-based optimization

Parallelism, such as multi-threading, I/O batching and so on, is an optimization technique to improve I/O performance, reduce overhead and memory consumption of the controllers [11,37,61,62]. These help increase controler's performance and therefore improve the scalability of the control plane.

Maestro [61] uses a multi-core architecture to leverage the parallelism in order to increase controller speed along with a hassle-free programming model for application writers. Maestro uses the batching of packets to individual destinations to improve processing and communication efficiency besides multi-threading structure. It is designed to evenly partition the workload in cores to increase the performance (i.e throughput) by keeping all processor cores busy by means of the "pull" fashion instead of the "push" fashion. It is pointed out that Maestro can achieve 600K requests/sec which implies that a distributed architecture of Maestro is needed to meet today's data center requirements.

McNettle [62] exploits multi-core opportunities of the Glasgow Haskell Compiler (GHC)[5] [78] and the run-time system. A certain number of CPU cores supports the McNettle system to scale up and the control algorithms requiring a global network state of flow arrival rates. In McNettle, when a packet cannot be associated with a flow rule, a packet-miss message is sent by a corresponding switch to invoke the packet-miss function included in message handlers forming McNettle programs. The authors claim that McNettle may scale up to 5K switches with 46 cores over a single controller with up to 13M flows/sec.

NOX-MT [37], the successor of NOX, is also a multi-thread controller which surpasses its predecessor (NOX) regarding throughput and response time. It embodies the fact that performance of a controller can be improved to certain levels by exploiting some optimization techniques such as multi-threading, I/O batching, malloc implementations etc. The authors leverage a performance measurement benchmark, Cbench [79], to emulate the switches and compare results of three different controllers, Beacon, Maestro and NOX, with NOX-MT regarding controller responsiveness, throughput performance and controller latency. The NOX-MT outperforms the other controllers by handling 1.8M flow requests/sec with an average response time of 2 ms.

Erickson [11] reveals "Beacon" that provides an easy-to-handle environment for programmers, extra abilities to manage applications, and better performance. One incentive design decision behind the Beacon is to enable network operators/administrators in order to manage (adding and/or removing) applications while running the Beacon. The Beacon is reinforced for a high performance by multi-threaded designs: "Shared Queue" and "Run-To-Completion". In "Shared Queue" design, the pipeline threads take the messages from the shared queue in order to process by corresponding applications. In case of the "Run-To-Completion" design, on the other hand, there are no pipeline threads and each message is processed by I/O threads. The evaluation results show that the Beacon outperforms some other controllers such as Maestro [61], NOX etc. by responding 1.35M messages/sec with a single thread. It also scales linearly with 12 threads by responding more than 12.8M messages/sec.

6.2. Control plane routing scheme-based optimization

Reducing the processed events resulting from routing decisions of the controller(s) is another way to increase the scalability and performance of the control plane in an SDN architecture. In [71,72,80] the authors aim for a better and less event-producing routing schemes managed directly by controller(s) in order to scale up the OpenFlow-based networks.

Gao et al. [80] leverage a Dynamically Reconfigurable Processor (DRP) to increase the scalability of the controller. The authors exploit an emulated network-on-chip, called "diorama network", to perform routing. In the diorama network, they send emulated packets from source nodes to destination nodes through the network in order to figure out the shortest path. Their study is motivated by the fact that since routing by controllers affects the performance of controllers, slow routing decisions will increase the response time of controllers to switches in the data plane. An issue that is not investigated by the authors is how the proposed design copes with link failures in the network.

Source routing and its variations are also utilized to increase controller scalability and performance in SDN [71,72]. The underlying motivation behind these studies relies on reducing the state distributed by the controller to data plane devices. This state distribution on each switch on a path takes a long time and pushes the controller response time. Hence, it increases the delay and network convergence time. It exploits the idea of inserting path information in packet headers so that each node can acquire the next node information where the packets are to be sent without communicating with the controller. This approach is different from the traditional OpenFlow hop-by-hop routing model in which each node

[5] GHC is an open source compiler for Haskell [77] (a functional programming language).

communicates to the controller to learn what to do and where to send the flows.

QuagFlow [81] and RouteFlow [82] (evaluation of the QuagFlow) are some other projects that aim at certain objectives: (1) utilization of cheap network devices with minimal embedded software, (2) enabling use of legacy IP routing protocols, OSPF, RIP, BGP etc., without re-writing in a centralized way, and (3) ensuring interoperability with legacy network devices. They provide a transparent unification of the Quagga routing software suite [83] and OpenFlow-enabled hardware. They run control logic of underlying OpenFlow switches through a virtual network composed by virtual machines (VMs), which execute a routing engine (e.g. Quagga). These VMs are connected to each other to represent the physical topology. The virtual environment is kept in external servers communicating with a controller application. Decisions made by the legacy IP protocols are converted to flow rules by the controller application and installed to switch flow tables by the controller. Therefore, there is no requirement for modification of the existing routing protocols.

Scalability in carrier-grade networks also requires attentions from researchers due to some reasons such as number of and geographical distances between devices. Hartert et al. [49] propose a solution framework, DEFO (Declarative and Expressive Forwarding Optimizer), to achieve high scalability as well as robustness at carrier-grade networks. Their solution is based on two logical layers: connectivity layer and optimization layer. While the connectivity layer is responsible for default forwarding behavior and defines connectivity paths, the optimization layer defines exceptions to this default forwarding behavior and implements optimized paths, which are overwritten connectivity paths and computed by stitching connectivity paths together.

7. Comparison of control plane scalability proposals

Controllers are the main entities in decision-making processes in SDN networks. They perform crucial tasks affecting performance of the whole network. Currently, there exist more than 35 different publicly-available and proprietary SDN OpenFlow controllers created by different research groups, vendors, and organizations from both academia and industry, written in different languages, and having different performances. This rapid growing of controllers has raised questions regarding performance benchmarking of these controllers. Some research efforts [45,84] have been proposed to evaluate performances of the controllers with respect to certain metrics. In [45], the authors present a limited analysis of controllers' performances by using a new benchmarking framework called "hcprobe". Similarly, Jarschel et al. [84] also introduce a tool called "OFCBenchmark" to benchmark OpenFlow controllers. As stated earlier, the performance of an SDN controller is characterized by several metrics, but, throughput and flow setup latency are the most considered ones by the SDN research community. In terms of the control plane scalability, the *throughput* metric typically represents the number of flows that a control plane (i.e. controller) can handle in certain amount of time while the *flow setup latency* denotes the time elapsed from arrival of a "packet_in" message from a switch to installation of the corresponding flow rule in the switch flow table.

Table 2 shows performance related results of studies with respect to some scalability related metrics such as throughput and flow setup latency.

Since some studies evaluate performance of their systems regarding different metrics such as path installation time [67], ratio of elephant to mouse flows [59], link utilization [57] and so on, it is difficult to show all the metrics used in studies in a table. In addition, we note that these numbers heavily depend on the evaluation environments. In other words, each study uses different network dynamics and parameters such as workload, network topology, number of controllers, applications for testing etc. during their experiments. Also, these controllers are designed for different problems. Therefore, we highly recommend readers to individually examine the corresponding studies in order to rightly evaluate their scalability performances with respect to their characteristics.

Table 2
Scalability related some metrics such as control plane throughput in terms of the number of flows handled and flow setup latency by control plane from the studies.

Proposals \ metrics	Throughput (Flows/sec)	Flow Setup latency
Beacon [11]	up to 12.8M	avg 24.7 µs
DevoFlow [50]	–	–
DIFANE [51]	up to 3M	min 0.4 ms
DISCO [52]	–	–
D-SDN [53]	–	–
ElastiCon [54]	up to 30K	min 1 ms
Ethane [55]	up to 11K	min 1.5 ms
Fibbing [56]	–	min 0.89 ms
FlowBroker [57]	–	–
HyperFlow [58]	–	–
Kandoo [59]	up to 1.3M	–
Logical xBar [60]	–	–
Maestro [61]	up to 3.5M	avg 55 ms
McNettle [62]	up to 13M	max 10 ms
NOX [9]	up to 30K	avg 49 ms
NOX-MT [37]	up to 1.8M	avg 2 ms
Onix [63]	up to 200K	min 2 ms
ONOS [64]	up to 19K	avg 34 ms
Orion [65]	up to 50K	min 11 ms
Tam et al. [67]	–	–
Yazici et al. [68]	up to 36K	–
Bari et al. [69]	–	min 5 ms
Karakus et al. [70]	–	–
Owens et al. [71]	–	–
Soliman et al. [72]	–	–

Using different number of threads shows that single threaded controllers, such as Ethane and NOX, are very limited regarding the throughput because they cannot handle a large number of flows. However, the controllers that are multi-threaded, such as Beacon, Maestro, McNettle, and NOX-MT, can handle a large number of flows per second. The authors in [68] report that the average number of controller responses per second per switch when one, two, three, and four controllers are used are approximately 6K, 12K, 25K, and 36K, respectively. ElastiCon's throughput performance with respect to the number of controllers varies from 30K to 72K, its response time performance for packet-in arrivals up to 2K packets/sec regarding 1-controller, 2-controllers, and 4-controllers cases varies from 1.1 ms to 13.8 ms, 1.0 ms to 4.3 ms, 1.0 ms to 2.2 ms, respectively. In Orion architecture, the total number of new flows that area controller(s) can handle per second varies from 8K to around 50K with respect to the number of area controllers. It is also reported that minimum average flow setup delay between areas is around 11 ms while maximum of which reaches to around 25 ms depending on the number of domain controllers, areas, and switches in an area. In [69], the authors state that their framework shows around 160 ms and 5 ms average flow setup time performance for 1-controller and n-controllers cases, respectively, on RF-I topology (79 nodes, 294 links) while it is 185 ms and 12 ms, respectively, on RF-II topology (108 nodes, 306 links). In ONOS, 45.2 ms and 34.1 ms latency values are reported for the time elapsed from a network event detection to sending first corresponding `OFPT_FLOW_MOD` message for rerouting 1K flows and path installation, respectively. In DIFANE, packets experience 0.4 ms round-trip time at 100 single-packet flows/sec sending rate. While NOX-MT has an average response time of 2 ms, Beacon has the minimum average latency with 24.7 µs among the others.

Table 3
Some features such as the controller that works with, used programming language and evaluation setup characteristics of the studies.

Proposals \ features	controller	Programming language	Evaluation setup
Beacon [11]	Beacon	Java	Used Cbench for tests run on Amazon's Elastic Computer Cloud using a Cluster Compute Eight Extra Large instance containing 16 cores.
DEFO [49]	DEFO	Scala	Used many different real and realistic topologies with different number of nodes and links and compared it to Cisco MATE [85], a traffic engineering tool.
DevoFlow [50]	Any	Depends on controller used	Implemented a flow-level data center network simulator. Used a three-level Clos topology w/ 168 switches and a two-dimensional HyperX topology w/ 97 switches.
DIFANE [51]	Any	Depends on controller used	Used XORP [86] to run the link-state routing protocol and kernel-level Click-based OpenFlow switches as a authority switches.
DISCO [52]	Any	DISCO	Used Floodlight [87] controllers and Mininet [88] SDN simulator to create 3 SDN WANs w/ 4 switches each and connected to each other.
DRP [80]	Any	Depends on controller used	Constructed an emulated network (w/ 6 routers and 10 links) on a commercially available dynamically reconfigurable processor DAPDNA-2.
ElastiCon [54]	Any	Java	Used modified Floodlight controller, k = 4 fat tree topology and a modified version on Mininet to run the Open vSwitch [89] instances on different hosts.
Ethane [55]	Ethane Controller	C++/Python	Deployed at Stanford's Computer Science department for over 4 months and managed over 19 switches and 300 hosts.
Fibbing [56]	Fibbing Controller	Python/C	Used ISP topologies [90] whose sizes range from 80 nodes to over 300. All measurements were performed using OSPF on a Cisco ASR9K router equipped with 12GB of DRAM as well as on a Juniper M120 router equipped with 2GB of DRAM.
FlowBroker [57]	Any	Java	Used Floodlight controller and Mininet tool to test 5 different scenarios.
HyperFlow [58]	NOX	C++	Used 10 servers each equipped with a gigabit NIC and running as a storage node.
Kandoo [59]	Kandoo	C/C++/Python	Used a simple tree topology where each switch is controlled by one local controller and kandoo root controller atop in modified version of Mininet and Open vSwitch.
Maestro [61]	Maestro	Java	Implemented an emulator to generate flow requests from hosts on a common 79-switch topology going to Maestro controller.
McNettle [62]	McNettle	Haskell	Used a modified version of Cbench and ran the controller on a DELL Poweredge R815 server with 48 cores.
NOX [9]	NOX	C++/Python	Ran it in their internal network of roughly 30 hosts for over 6 months.
NOX-MT [37]	NOX-MT	C++/Python	Used Cbench representing 100K hosts and 32 emulated switches.
Onix [63]	Onix	C++	Evaluated Onix in two ways: with micro-benchmarks to test Onix's performance as a general platform, and with end-to-end performance measurements of an in-development Onix application in a test environment.
ONOS [64]	Any	Java	Used Floodlight controller and connected a 6-node ONOS cluster to an emulated Mininet network of 206 software switches and 416 links. Also demonstrated in Internet2 [91] topology.
Orion [65]	Any	Java	Used Floodlight controller as area controllers. Conducted different experiments for different number of domain (from 1 to 2) and area controllers (from 1 to 6) and switches (from 20 to 120).
Tam et al. [67]	Any	Depends on controller used	Used 4 controllers on topology of an irregular network with 28 nodes and 66 links.
Yazici et al. [68]	Any	Java	Used Beacon controllers for the experimental setup with 4 controllers and 4 emulated switches to run Cbench instances.
Bari et al. [69]	Any	Python	Used POX [92] controller and Mininet to simulate RF-I (79 nodes, 294 links) and RF-II (108 nodes, 306 links) ISP topologies.
Karakus et al.[70]	Any	Depends on controller used	Used a topology with 4 different autonomous domains with 4 switches each and a Broker connected to domain controllers.
Owens et al. [71]	VSDN Controller	C/C++	Used NS-3 [93] tool to simulate a 6-node network with increasing connection requests for the controller.
Soliman et al. [72]	Any	Depends on controller used	Used Internet2 OS3E topology with 34-nodes.

Table 3 illustrates some features, such as the controller that works with, used programming language in controller implementation and evaluation setup characteristics, of the studies. Most of the proposals work with any SDN controller with some modification efforts. However, the Floodlight [87] controller is the most used one in the evaluation phase of the studies due to its good documentation, active community support, and integration with REST API. Also, Java is the prevalent programming language used in implementation of the studies due to their controller choice although some studies do not report which programming language they used.

Table 4 illustrates the approaches used by the studies to achieve control plane scalability. Topology-related approaches uses single, distributed (flat) or hierarchical controller designs. Mechanisms-related approaches exploit multi-threading, I/O batching, better routing schemes etc. There are also hybrid (i.e. both the control plane-centric and data plane-centric) studies. We note that some research efforts belong to more than one approach because they more or less exploit some other approaches too. However, they have been classified based on their main methods, which are discussed in corresponding sections.

Controller designers may consider two architectural design goals while designing their controllers to improve scalability performance: (1) they can utilize static *switch partitioning*—distribution and allocation of connected network devices to worker threads running in the controller—and *packet batching*—where multiple bytes are read from or written to the underlying network using a socket buffer—techniques to achieve high throughput and (2) workload adaptive *packet batching* and *task batching*—a strategy used to allocate already received packets to the worker threads for processing, hence directly impacting the latency of the controller—to reduce flow setup latency.

8. Challenges and existing proposals in SDN control plane

While SDN is becoming a mature technology, the control plane scalability issues deserve more research efforts from both academia and industry. In this section, we discuss the general problems in an SDN control plane. However, each of these problems affects the scalability of the control plane in SDN. Therefore, these problems need to be taken care of by network operators while designing/operating their SDN networks. In the following, we

Table 4
Approaches used by the studies to achieve control plane scalability. Topology-related approaches utilizes central (single), distributed (flat), hierarchical controller and hybrid designs. Mechanisms-related approaches exploit multi-threading, I/O batching, better routing schemes etc. We note that some research efforts belong to more than one approach because they exploit some other approaches in their designs too. However, they have been classified based on their primary approaches, which are discussed in the corresponding (sub)sections.

Proposals \ approaches	Topology-related approaches				Mechanisms-related approaches	
	Centralized (Single) Controller Designs	Distributed Approaches			Parallelism-based Optimization	Control Plane Routing-based Optimization
		Distributed (Flat) Controller Designs	Hierarchical Controller Designs	Hybrid Designs		
Beacon [11]					✓	
DEFO [49]		✓				✓
DevoFlow [50]				✓	✓	
DIFANE [51]				✓		
DISCO [52]		✓				
DRP [80]					✓	✓
D-SDN [53]			✓			
ElastiCon [54]		✓				
Ethane [55]	✓					
Fibbing [56]	✓	✓		✓		
FlowBroker [57]			✓			
HyperFlow [58]		✓			✓	
Kandoo [59]			✓			
Logical xBar [60]			✓			
Maestro [61]					✓	
McNettle [62]					✓	
NOX [9]	✓				✓	
NOX-MT [37]	✓				✓	
Onix [63]		✓				
ONOS [64]		✓				
Orion [65]			✓			
Tavakoli et al. [66]	✓					
Tam et al. [67]		✓			✓	
Yazici et al. [68]		✓			✓	
Bari et al. [69]		✓				
Karakus et al. [70]			✓			✓
Owens et al. [71]						✓
Soliman et al. [72]						✓

state the main SDN control plane challenges along with existing proposals.

- ***Controller(s) failure:*** In a traditional network, when one or more network nodes fail, flows are routed through alternative paths/nodes to maintain the traffic continuity. However, in an SDN architecture, failure of the controller(s) may result in a chaos for the specific part(s) of the network controlled by the failed controller(s) due to two main critical reasons: (i) The controllers are responsible for all configurations, operations, and validations of the network topologies, resources etc. and (ii) data plane devices lack an ability for an online "detour" of flows. This problem may become worse in the single controller design case. In addition, distributing the load of a failed controller to other controllers brings extra load on them, which reduces performances thereby their scalability. This distribution may even result in a cascading failures of controllers because it can exceed the capacity of them.
  One way to address this problem is to enhance the network with backup/standby controllers [94,95]. In case of the main controller failure, these backup controller(s) may take the responsibility of the network operations over from the main controller. In this case, controllers need to be synchronized to be in a consistent status regarding network states.
  In [96], the authors present a disaster-aware control plane design to reduce controller-related interruptions. They model the problem of designing a disaster-resilient control plane problem regarding the number of controllers, their placement, and the control plane topology. Pashkov et al. [97] propose a fault-tolerant control plane design, High-Available Controller (HAC) architecture, to address the fast recovery of the control plane by adding an additional cluster middleware between the controller core and controller network services and applications.
- ***State/policy distribution/consistency:*** Another important problem regarding scalability is the network state distribution and consistency between controllers of a control plane. This problem mainly happens in the distributed and/or hierarchical architectures due to distribution of network states among controller replicas. In addition, this distribution needs to be fast and reliable to provide the consistency between controller instances [98]. Moreover, policy consistency [73] in a distributed control plane is required because network-wide policies do not come from a single component of a network, but rather, they are formed by different functional modules such as routing, monitoring, and access control as well as multiple human operators controlling different parts of the network. These conflicts may result in serious inconsistencies such as violation to another policy and wrong forwarding of the packet etc. on the data plane. Therefore, more efficient algorithms and mechanisms are needed to maintain state/policy consistency among the distributed controllers.
  Distributing network state among local controllers in the same domain does not necessarily deal with security issues. However, the Internet consists of many networks managed by different authorities. Therefore, the logically centralized control model of SDN must be extended to account for inter-domain traffic. This extension requires peering, thereby state sharing, among different administrative domains to have a relative global network view in order to determine the next hop. However, this distribution has to be secure, private, and consistent. In addition, some other critical questions regarding this sharing are how

and what to exchange with other domains. Yin et al. [99] state that the types of messages exchanged among controllers may be various such as reachability information, flow setup/teardown/update requests, network parameters (bandwidth, delay, loss etc.), service-level agreements (SLAs), virtual network information and so on. In [25,100,101], the authors propose a West-East (WE) Bridge mechanism to enable different SDN administrative domains to securely peer and cooperate with each other.

- ***Flow rule setup latency:*** This problem refers to the delay in new flow rule setup process in the context of control plane scalability [40]. As explained earlier, proactive mode and reactive mode are two prominent modes to setup a new flow rule. The proactive mode does not impose any latency in the flow rule setup from the controller's point of view. In the reactive mode, the controller response time (i.e. delay) is crucial. Controllers having longer flow rule setup latencies may not meet requirements of certain applications such as fast fail-over and reactive routing of latency-sensitive flows. Therefore, such control planes cannot be scalable enough to satisfy the network needs. However, this delay can be relatively reduced by imposing more controller and switch resources such as CPU, memory etc. and devolving some control functions to the switches.
  In [102], the authors conduct various setup experiments to test the latency of various controllers by changing the number of switches, number of threads, and controller workload. They conclude that adding more threads beyond the number of switches does not improve latency, and serving more switches than available CPUs increases controller response time.
  Some studies [50,51] mitigate the flow rule setup latency by leveraging the idea of *Control Function Devolvement*. This idea relies on the delegation of some of the control functions to the data plane so as to alleviate the load on the controller(s), thereby reducing controller-switch communication frequency.
- ***Controller placement:*** In addition the number of controllers, placement of the controller(s) [47] has impacts on performance of the network as well. Suboptimal controller placement affects many other problems such as flow rule setup latency due to controller-switch communication delay, controller-controller communication delay, control plane overhead, fault tolerance, resiliency and so on. Although there are some studies addressing this problem in the literature [103–110], we believe it is still an ongoing issue and needs further attention of researchers.
  Hu et al. [103] propose algorithms to automate the controller placement decisions given a physical network and the number of controllers. The main objective of these algorithms is to maximize resiliency of SDN to failures. In [104,105], the authors address the controller placement problem to maximize the reliability of control networks.
  Rath et al. [106] present a non-zero-sum game-based distributed technique to discuss optimal controller placement in SDN. Hock et al. [107] introduce the Pareto-based Optimal COntroller-placement (POCO) framework, which brings network operators a range of options to select the controller placement based on their particular needs regarding the metrics like latency, load balancing etc. In [108], the authors focus on the controller placement problem for WAN. They use the Spectral Clustering placement algorithm to partition a large network into several small SDN domains. Jimenez et al. [109] utilize the algorithm called "k-Critical" to discover the minimum number of controllers and their locations to create a robust control topology that deals with failures and balances the load among the selected controllers.
  Furthermore, control plane overhead is affected by the placement of controllers due to traffic between switches and controllers (*packet_in* and *flow_mod* messages) and among controllers (e.g. state sharing). Obadia et al. [110] challenge the problem of minimizing control overhead by optimizing the number of controllers and their placement. This approach differs from others because they target minimization of control overhead instead of minimization of switch-controller delay.

9. Conclusions

Software Defined Networking is a promising emerging architecture for many networking environments such as data centers, enterprise networks, campus networks, cloud networks, and WAN. The major advantages of SDN are its programmability and agility. However, the scalability issues in the control plane is one major problem in SDN that needs more research attention. In the paper, we have firstly given an overview of the SDN architecture and OpenFlow protocol along with its support mechanisms for scalability. We have discussed the scalability as a concept in general and presented various metrics proposed for quantification of scalability. We have seen that there is no consensus on the definition of scalability. In other words, while the basic notion is intuitive, scalability does not evoke the same concept to everybody. In the context of SDN, scalability is characterized by the two prominent metrics, throughput and flow setup latency. Also, we have pointed out the main reasons that make the control plane a scalability bottleneck in SDN: Separation of Control Plane and Data Plane, Quantity of Events/Requests Handled by a Controller, and Controller-Switch Communication Delay. Furthermore, we have presented our organization for taxonomy of scalability-centric studies in two broad approaches: Topology-related approaches and Mechanisms-related approaches. While the former reviews the relation between topology of architectures and scalability issues, the latter discusses the relation between various mechanisms used to optimize controllers and scalability issues. Finally, we have outlined the potential challenges and open problems that need to be addressed further for more scalable SDN control planes: Controller(s) Failure, State/Policy Distribution/Consistency, Flow Rule Setup Latency, and Controller Placement.

Acknowledgments

This work was partially supported by National Science Foundation under Grant No. 1547411.

References

[1] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, N. Rao, Are we ready for sdn? implementation challenges for software-defined networks, Commun. Mag., IEEE 51 (7) (2013) 36–43, doi:10.1109/MCOM.2013.6553676.
[2] Software-Defined Networking: The New Norm for Networks, Technical Report. Open Networking Foundation (ONF), 2012.
[3] K. Kirkpatrick, Software-defined networking, Commun. ACM 56 (9) (2013) 16–19.
[4] F. de Oliveira Silva, J. de Souza Pereira, P. Rosa, S. Kofuji, Enabling future internet architecture research and experimentation by using software defined networking, in: Software Defined Networking (EWSDN), 2012 European Workshop on, 2012, pp. 73–78, doi:10.1109/EWSDN.2012.24.
[5] G. Goth, Software-defined networking could shake up more than packets, Internet Comput., IEEE 15 (4) (2011) 6–9, doi:10.1109/MIC.2011.96.
[6] W. Xia, Y. Wen, C. Foh, D. Niyato, H. Xie, A survey on software-defined networking, Commun. Surv. Tut., IEEE PP (99) (2014). 1–1, doi: 10.1109/COMST.2014.2330903.
[7] K. Bakshi, Considerations for software defined networking (sdn): approaches and use cases, in: Aerospace Conference, 2013 IEEE, 2013, pp. 1–9, doi:10.1109/AERO.2013.6496914.
[8] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A. Vahdat, B4: experience with a globally-deployed software defined wan, in: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, in: SIGCOMM '13, ACM, New York, NY, USA, 2013, pp. 3–14, doi:10.1145/2486001.2486019.
[9] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker, Nox: towards an operating system for networks, SIGCOMM Comput. Commun. Rev. 38 (3) (2008) 105–110, doi:10.1145/1384609.1384625.

[10] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, R. Chaiken, The nature of data center traffic: Measurements & analysis, in: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference, in: IMC '09, ACM, New York, NY, USA, 2009, pp. 202–208, doi:10.1145/1644893.1644918.

[11] D. Erickson, The beacon openflow controller, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, in: HotSDN '13, 2013, pp. 13–18.

[12] Y. Luo, P. Cascon, E. Murray, J. Ortega, Accelerating openflow switching with network processors, in: Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, in: ANCS '09, ACM, New York, NY, USA, 2009, pp. 70–71, doi:10.1145/1882486.1882504.

[13] V. Tanyingyong, M. Hidell, P. Sjödin, Using hardware classification to improve pc-based openflow switching, in: 2011 IEEE 12th International Conference on High Performance Switching and Routing, IEEE, 2011, pp. 215–221.

[14] R. Narayanan, S. Kotha, G. Lin, A. Khan, S. Rizvi, W. Javed, H. Khan, S.A. Khayam, Macroflows and microflows: Enabling rapid network innovation through a split sdn data plane, in: 2012 European Workshop on Software Defined Networking, 2012, pp. 79–84, doi:10.1109/EWSDN.2012.16.

[15] G. Lu, R. Miao, Y. Xiong, C. Guo, Using cpu as a traffic co-processing unit in commodity switches, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, in: HotSDN '12, ACM, New York, NY, USA, 2012, pp. 31–36, doi:10.1145/2342441.2342448.

[16] K. Kannan, S. Banerjee, Compact TCAM: Flow Entry Compaction in TCAM for Power Aware SDN, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 439–444. doi:10.1007/978-3-642-35668-1_32

[17] N. Kang, Z. Liu, J. Rexford, D. Walker, Optimizing the "one big switch" abstraction in software-defined networks, in: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, in: CoNEXT '13, ACM, New York, NY, USA, 2013, pp. 13–24, doi:10.1145/2535372.2535373.

[18] N. Katta, O. Alipourfard, J. Rexford, D. Walker, Infinite cacheflow in software-defined networks, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, in: HotSDN '14, ACM, New York, NY, USA, 2014, pp. 175–180, doi:10.1145/2620728.2620734.

[19] Q. Zuo, M. Chen, K. Ding, B. Xu, On generality of the data plane and scalability of the control plane in software-defined networking, China Commun. 11 (2) (2014) 55–64, doi:10.1109/CC.2014.6821737.

[20] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.

[21] F. Hu, Q. Hao, K. Bao, A survey on software defined networking (sdn) and openflow: from concept to implementation, Commun. Surv. Tut., IEEE PP (99) (2014). 1–1. doi: 10.1109/COMST.2014.2326417.

[22] S.J. Vaughan-Nichols, Openflow: the next generation of the network? IEEE Comput. 44 (8) (2011) 13–15.

[23] OpenFlow Switch Specification, 1.5.1, 2015. URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf

[24] SDN architecture. Technical Report. Open Networking Foundation (ONF). 2014.

[25] P. Lin, J. Bi, S. Wolff, Y. Wang, A. Xu, Z. Chen, H. Hu, Y. Lin, A west-east bridge based sdn inter-domain testbed, Commun. Mag., IEEE 53 (2) (2015) 190–197, doi:10.1109/MCOM.2015.7045408.

[26] M. Fernandez, Comparing OpenFlow controller paradigms scalability: reactive and proactive, in: Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on, 2013, pp. 1009–1016, doi:10.1109/AINA.2013.113.

[27] OpenFlow Switch Specification, 1.1.0, 2011. URL http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf.

[28] OpenFlow Switch Specification, 1.2.0, 2011. URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf.

[29] B.J. van Asten, N.L.M. van Adrichem, F.A. Kuipers, Scalability and resilience of software-defined networking: an overview, CoRR (2014). 1408.6760.

[30] K. Hwang, Z. Xu, Scalable Parallel Computing: Technology, Architecture, Programming, McGraw-Hill, Inc., New York, NY, USA, 1998.

[31] X.H. Sun, D.T. Rover, Scalability of parallel algorithm-machine combinations, IEEE Trans. Parallel Distrib. Syst. 5 (6) (1994) 599–613, doi:10.1109/71.285606.

[32] V. Kumar, A. Grama, A. Gupta, G. Karypis, Introduction to Parallel Computing: Design and Analysis of Algorithms, Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994.

[33] A.Y. Grama, A. Gupta, V. Kumar, Isoefficiency: measuring the scalability of parallel algorithms and architectures, IEEE Concurrency (3) (1993) 12–21.

[34] L. Pastor, J. Bosquwe Orero, An efficiency and scalability model for heterogeneous clusters, in: Cluster Computing, 2001. Proceedings. 2001 IEEE International Conference on, 2001, pp. 427–434, doi:10.1109/CLUSTR.2001.960009.

[35] X.H. Sun, Y. Chen, M. Wu, Scalability of heterogeneous computing, in: Proceedings of the 2005 International Conference on Parallel Processing, in: ICPP '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 557–564, doi:10.1109/ICPP.2005.69.

[36] P. Jogalekar, M. Woodside, Evaluating the scalability of distributed systems, IEEE Trans. Parallel Distrib. Syst. 11 (6) (2000) 589–603, doi:10.1109/71.862209.

[37] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, R. Sherwood, On controller performance in software-defined networks, in: Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, in: Hot-ICE'12, 2012. 10–10.

[38] S.A. Shah, J. Faiz, M. Farooq, A. Shafi, S.A. Mehdi, An architectural evaluation of sdn controllers, in: 2013 IEEE International Conference on Communications (ICC), 2013, pp. 3504–3508, doi:10.1109/ICC.2013.6655093.

[39] F. Benamrane, R. Benaini, et al., Performances of openflow-based software-defined networks: an overview, Journal of Networks 10 (6) (2015) 329–337.

[40] K. He, J. Khalid, S. Das, A. Gember-Jacobson, C. Prakash, A. Akella, L.E. Li, M. Thottan, Latency in software defined networks: Measurements and mitigation techniques, in: Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, in: SIGMETRICS '15, ACM, New York, NY, USA, 2015, pp. 435–436, doi:10.1145/2745844.2745880.

[41] P. Isaia, L. Guan, Performance benchmarking of sdn experimental platforms, in: 2016 IEEE NetSoft Conference and Workshops (NetSoft), 2016, pp. 116–120, doi:10.1109/NETSOFT.2016.7502456.

[42] J. Hu, C. Lin, X. Li, J. Huang, Scalability of control planes for software defined networks: Modeling and evaluation, in: Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of, 2014, pp. 147–152, doi:10.1109/IWQoS.2014.6914314.

[43] M. Karakus, A. Durresi, A scalability metric for control planes in Software Defined Networks (SDNs), in: 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), 2016, pp. 282–289, doi:10.1109/AINA.2016.112.

[44] S. Yeganeh, A. Tootoonchian, Y. Ganjali, On scalability of software-defined networking, Commun. Mag., IEEE 51 (2) (2013) 136–141, doi:10.1109/MCOM.2013.6461198.

[45] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, R. Smeliansky, Advanced study of sdn/openflow controllers, in: Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, in: CEE-SECR '13, ACM, New York, NY, USA, 2013, pp. 1:1–1:6, doi:10.1145/2556610.2556621.

[46] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, in: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, in: IMC '10, ACM, New York, NY, USA, 2010, pp. 267–280, doi:10.1145/1879141.1879175.

[47] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, in: Proceedings of the first workshop on Hot topics in software defined networks, in: HotSDN '12, 2012, pp. 7–12.

[48] S. Azodolmolky, P. Wieder, R. Yahyapour, Performance evaluation of a scalable software-defined networking deployment, in: Software Defined Networks (EWSDN), 2013 Second European Workshop on, 2013, pp. 68–74, doi:10.1109/EWSDN.2013.18.

[49] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, P. Francois, A declarative and expressive approach to control forwarding paths in carrier-Grade networks, SIGCOMM Comput. Commun. Rev. 45 (4) (2015) 15–28, doi:10.1145/2829988.2787495.

[50] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, Devoflow: scaling flow management for high-performance networks, SIGCOMM Comput. Commun. Rev. 41 (4) (2011) 254–265, doi:10.1145/2043164.2018466.

[51] M. Yu, J. Rexford, M.J. Freedman, J. Wang, Scalable flow-based networking with difane, SIGCOMM Comput. Commun. Rev. 41 (4) (2010). –

[52] K. Phemius, M. Bouet, J. Leguay, Disco: distributed multi-domain sdn controllers., CoRR abs/1308.6138 (2013).

[53] M. Santos, B. Nunes, K. Obraczka, T. Turletti, B. de Oliveira, C. Margi, Decentralizing sdn's control plane, in: Local Computer Networks (LCN), 2014 IEEE 39th Conference on, 2014, pp. 402–405, doi:10.1109/LCN.2014.6925802.

[54] A.A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, R. Kompella, Elasticon: an elastic distributed sdn controller, in: Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems, ACM, 2014, pp. 17–28.

[55] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, Ethane: taking control of the enterprise, SIGCOMM Comput. Commun. Rev. 37 (4) (2007) 1–12.

[56] S. Vissicchio, O. Tilmans, L. Vanbever, J. Rexford, Central control over distributed routing, SIGCOMM Comput. Commun. Rev. 45 (4) (2015) 43–56, doi:10.1145/2829988.2787497.

[57] D. Marconett, S. Yoo, Flowbroker: a software-defined network controller architecture for multi-domain brokering and reputation, J. Netw. Syst. Manag. 23 (2) (2015) 328–359, doi:10.1007/s10922-014-9325-5.

[58] A. Tootoonchian, Y. Ganjali, Hyperflow: A distributed control plane for openflow, in: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, in: INM/WREN'10, 2010. 3–3.

[59] S. Hassas Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: Proceedings of the first workshop on Hot topics in software defined networks, in: HotSDN '12, 2012, pp. 19–24.

[60] J. Mccauley, A. P, M. Casado, T. Koponen, S. Shenker, Extending sdn to large-scale networks, ONS, 2013.

[61] Z. Cai, A.L. Cox, T.S.E. Ng, Maestro: A System for Scalable OpenFlow Control, Technical Report, Rice University, 2010.

[62] A. Voellmy, J. Wang, Scalable software defined network controllers, in: Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication, in: SIGCOMM '12, ACM, New York, NY, USA, 2012, pp. 289–290, doi:10.1145/2342356.2342414.

[63] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, S. Shenker, Onix: a distributed control platform for large-scale production networks, in: Proceedings of the 9th USENIX conference on Operating systems design and implementation, in: OSDI'10, 2010, pp. 1–6.
[64] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, G. Parulkar, Onos: towards an open, distributed sdn os, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, in: HotSDN '14, ACM, New York, NY, USA, 2014, pp. 1–6, doi:10.1145/2620728.2620744.
[65] Y. Fu, J. Bi, K. Gao, Z. Chen, J. Wu, B. Hao, Orion: a hybrid hierarchical control plane of software-defined networking for large-scale networks, in: Network Protocols (ICNP), 2014 IEEE 22nd International Conference on, 2014, pp. 569–576, doi:10.1109/ICNP.2014.91.
[66] A. Tavakoli, M. Casado, T. Koponen, S. Shenker, Applying nox to the datacenter, in: Proc. of workshop on Hot Topics in Networks (HotNets-VIII), 2009.
[67] A.-W. Tam, K. Xi, H. Chao, Use of devolved controllers in data center networks, in: Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on, 2011, pp. 596–601, doi:10.1109/INFCOMW.2011.5928883.
[68] V. Yazici, O.M. Sunay, A.O. Ercan, Controlling a software-defined network via distributed controllers, in: 2012 NEM Summit Conference Proceedings, in: NEM Summit'12, 2012.
[69] M. Bari, A. Roy, S. Chowdhury, Q. Zhang, M. Zhani, R. Ahmed, R. Boutaba, Dynamic controller provisioning in software defined networks, in: Network and Service Management (CNSM), 2013 9th International Conference on, 2013, pp. 18–25, doi:10.1109/CNSM.2013.6727805.
[70] M. Karakus, A. Durresi, A scalable inter-as qos routing architecture in software defined network (sdn), in: Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on, 2015, pp. 148–154, doi:10.1109/AINA.2015.179.
[71] H. Owens, A. Durresi, Explicit routing in software-defined networking (ersdn): addressing controller scalability, in: Network-Based Information Systems (NBiS), 2014 17th International Conference on, 2014.
[72] M. Soliman, B. Nandy, I. Lambadaris, P. Ashwood-Smith, Source routed forwarding with software defined control, considerations and implications, in: Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop, in: CoNEXT Student '12, 2012, pp. 43–44.
[73] M. Canini, P. Kuznetsov, D. Levin, S. Schmid, Software transactional networking: concurrent and consistent policy composition, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, in: HotSDN '13, ACM, New York, NY, USA, 2013, pp. 1–6, doi:10.1145/2491185.2491200.
[74] Opendaylight. URL https://www.opendaylight.org/.
[75] L.J. Cowen, Compact routing with minimum stretch, in: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, in: SODA '99, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999, pp. 255–260.
[76] S. Vissicchio, L. Vanbever, J. Rexford, Sweet little lies: fake topologies for flexible routing, in: Proceedings of the 13th ACM Workshop on Hot Topics in Networks, in: HotNets-XIII, ACM, New York, NY, USA, 2014, pp. 3:1–3:7, doi:10.1145/2670518.2673868.
[77] Haskell, a functional programming language. URL https://www.haskell.org/.
[78] The Glasgow Haskell compiler. URL https://www.haskell.org/ghc/.
[79] R. Sherwood, K.-K. Yap, Cbench: an openflow controller benchmarker. URL http://archive.openflow.org/wk/index.php/Oflops.
[80] S. Gao, S. Shimizu, S. Okamoto, N. Yamanaka, A high-speed routing engine for software defined network, Cyber J. (1) (2012) 1–7. JSAT August Edition.
[81] M.R. Nascimento, C.E. Rothenberg, M.R. Salvador, M.F. Magalhães, Quagflow: partnering quagga with openflow, in: Proceedings of the ACM SIGCOMM 2010 Conference, in: SIGCOMM '10, ACM, New York, NY, USA, 2010, pp. 441–442, doi:10.1145/1851182.1851252.
[82] M.R. Nascimento, C.E. Rothenberg, M.R. Salvador, C.N.A. Corrêa, S.C. de Lucena, M.F. Magalhães, Virtual routers as a service: the routeflow approach leveraging software-defined networks, in: Proceedings of the 6th International Conference on Future Internet Technologies, in: CFI '11, ACM, New York, NY, USA, 2011, pp. 34–37, doi:10.1145/2002396.2002405.
[83] GNU Quagga project. URL http://www.nongnu.org/quagga/.
[84] M. Jarschel, F. Lehrieder, Z. Magyari, R. Pries, A Flexible OpenFlow-Controller Benchmark, in: 2012 European Workshop on Software Defined Networking, 2012, pp. 48–53, doi:10.1109/EWSDN.2012.15.
[85] Planning and Designing Networks with the Cisco MATE Portfolio, 2013, Cisco URL http://www.cisco.com/c/en/us/products/collateral/routers/wan-automation-engine/white_paper_c11-728866.pdf.
[86] M. Handley, O. Hodson, E. Kohler, XORP: an open platform for network research, SIGCOMM Comput. Commun. Rev. 33 (1) (2003) 53–57, doi:10.1145/774763.774771.
[87] Floodlight OpenFlow Controller. URL http://www.projectfloodlight.org/floodlight/.
[88] Mininet. URL http://mininet.org/.
[89] Openvswitch. URL http://openvswitch.org/.
[90] N. Spring, R. Mahajan, D. Wetherall, T. Anderson, Measuring isp topologies with rocketfuel, IEEE/ACM Trans. Netw. 12 (1) (2004) 2–16, doi:10.1109/TNET.2003.822655.
[91] Internet2. URL http://www.internet2.edu/.
[92] Pox. URL http://www.noxrepo.org/pox/about-pox/.
[93] The NS-3 network simulator. URL http://mininet.org/.
[94] P. Fonseca, R. Bennesby, E. Mota, A. Passito, A replication component for resilient openflow-based networking, in: Network Operations and Management Symposium (NOMS), 2012 IEEE, 2012, pp. 933–939, doi:10.1109/NOMS.2012.6212011.
[95] F. Botelho, A. Bessani, F. Ramos, P. Ferreira, On the design of practical fault-tolerant sdn controllers, in: Software Defined Networks (EWSDN), 2014 Third European Workshop on, 2014, pp. 73–78, doi:10.1109/EWSDN.2014.25.
[96] S. Sedef Savas, M. Tornatore, M. Farhan Habib, P. Chowdhury, B. Mukherjee, Disaster-resilient control plane design and mapping in software-defined networks, ArXiv e-prints (2015).
[97] V. Pashkov, A. Shalimov, R. Smeliansky, Controller failover for sdn enterprise networks, in: Science and Technology Conference (Modern Networking Technologies) (MoNeTeC), 2014 International, 2014, pp. 1–6, doi:10.1109/MoNeTeC.2014.6995594.
[98] D. Levin, A. Wundsam, B. Heller, N. Handigol, A. Feldmann, Logically centralized?: State distribution trade-offs in software defined networks, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, in: HotSDN '12, ACM, New York, NY, USA, 2012, pp. 1–6, doi:10.1145/2342441.2342443.
[99] H. Yin, H. Xie, T. Tsou, D.R. Lopez, P.A. Aranda, R. Sidi, SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains, Internet-Draft, Internet Engineering Task Force, 2012. Work in Progress.
[100] P. Lin, J. Bi, Y. Wang, Webridge: west-east bridge for distributed heterogeneous sdn noses peering, Secur. Commun. Netw. 8 (10) (2015) 1926–1942, doi:10.1002/sec.1030.
[101] P. Lin, J. Bi, Z. Chen, Y. Wang, H. Hu, A. Xu, We-bridge: west-east bridge for sdn inter-domain network peering, in: Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on, 2014, pp. 111–112, doi:10.1109/INFCOMW.2014.6849180.
[102] J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A.R. Curtis, S. Banerjee, Devoflow: Cost-effective flow management for high performance enterprise networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, in: Hotnets-IX, ACM, New York, NY, USA, 2010, pp. 1:1–1:6, doi:10.1145/1868447.1868448.
[103] Y. nan HU, W. dong WANG, X. yang GONG, X. rong QUE, S. duan CHENG, On the placement of controllers in software-defined networks, J. China Univ. Posts Telecommun. 19, Supplement 2 (2012) 92–171. http://dx.doi.org/10.1016/S1005-8885(11)60438-X.
[104] Y. Hu, W. Wendong, X. Gong, X. Que, C. Shiduan, Reliability-aware controller placement for software-defined networks, in: Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, 2013, pp. 672–675.
[105] Y. Hu, W. Wang, X. Gong, X. Que, S. Cheng, On reliability-optimized controller placement for software-defined networks, Commun., China 11 (2) (2014) 38–54, doi:10.1109/CC.2014.6821736.
[106] H. Rath, V. Revoori, S. Nadaf, A. Simha, Optimal controller placement in software defined networks (sdn) using a non-zero-sum game, in: World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a, 2014, pp. 1–6, doi:10.1109/WoWMoM.2014.6918987.
[107] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, P. Tran-Gia, Pareto-optimal resilient controller placement in sdn-based core networks, in: Teletraffic Congress (ITC), 2013 25th International, 2013, pp. 1–9, doi:10.1109/ITC.2013.6662939.
[108] P. Xiao, W. Qu, H. Qi, Z. Li, Y. Xu, The sdn controller placement problem for wan, in: Communications in China (ICCC), 2014 IEEE/CIC International Conference on, 2014, pp. 220–224, doi:10.1109/ICCChina.2014.7008275.
[109] Y. Jimenez, C. Cervello-Pastor, A. Garcia, On the controller placement for designing a distributed sdn control layer, in: Networking Conference, 2014 IFIP, 2014, pp. 1–9, doi:10.1109/IFIPNetworking.2014.6857117.
[110] M. Obadia, M. Bouet, J.L. Rougier, L. Iannone, A Greedy Approach for Minimizing SDN Control Overhead, in: Network Softwarization (NetSoft), 2015 1st IEEE Conference on, 2015, pp. 1–5, doi:10.1109/NETSOFT.2015.7116135.

Murat Karakus received the BS Degree in Mathematics from Suleyman Demirel University, Turkey, in 2009, and the MS Degree in Computer Science and Information Systems from the University of Michigan-Flint, in 2013. He is currently working through his PhD in the Department of Computer and Information Science at Indiana University Purdue University - Indianapolis. He is the recipient of the Best Paper Award at ACM SIGITE 2011 conference. His current research interests include new network architectures (particularly Software-Defined Networking (SDN)), scalability, Quality of Service (QoS), routing, and introducing programming to non-CS majors.

Arjan Durresi is a Professor of Computer Science at Indiana University Purdue University in Indianapolis, Indiana. In the past, he held positions at LSU, and The Ohio State University. His research interest include network architectures, security and trust management. He has published over eighty papers in journals and over 200 papers in conference proceedings, and seven book chapters. He also has over thirty contributions to standardization organizations such as IETF, ATM Forum, ITU, ANSI and TIA. His research has been funded by NSF, the States of Ohio and Louisiana, as well as university and industry sources.