Multi-Domain Software Defined Networking: Research Status and Challenges

Francisciscus X. A. Wibowo, Mark A. Gregory, Khandakar Ahmed, Karina M. Gomez
RMIT University
Melbourne, Australia

***Abstract*— A key focus of the transition to next generation computer networking is to improve management of network services thereby enhancing traffic control and flows while simplifying higher-level functionality. Software-defined networking (SDN) is an approach that is being developed to facilitate next generation computer networking by decoupling the traffic control system from the underlying traffic transmission system. SDN offers programmability in network services by separating the control plane from the data plane within network devices and providing programmability for network services. Enhanced connectivity services across the global digital network require a multi-domain capability. This paper presents a review of the current research status in SDN and multi-domain SDN, focusing on OpenFlow protocol, and its future related challenges.**

***Index Terms*— Software Defined Networking, Networking, OpenFlow, Controller, East-West, Multi-domain**

1. Introduction

Software Defined Networking (SDN) concepts are moving from the data centre to the enterprise networks and Wide Area Networks (WAN) presenting a number of challenges. Telecommunication network growth and complexity continues unabated and people and machines can communicate with each other through various network types utilizing a range of technologies. Global IP Traffic is predicted to increase from 59.9 Exabytes per month in 2014 to 168.4 Exabytes per month in 2019 (Cisco, 2015). Smartphone and tablet use accounted for 40% of the global IP traffic in 2014 and is expected to rise to 67% in 2019. This phenomenon has led to the need for a high capacity, reliable and yet cost-effective network that can carry increasing traffic volume, with dynamic and distinct applications for each entity.

The global digital networks are currently struggling to meet the increasing traffic volumes, and the shift from traditional enterprise networks to distributed Cloud Computing exacerbates this trend. The current network architectures were based on a vertically integrated approach with discrete semi-autonomous devices, which limit the potential for flexible flow management and network service innovation.

Service and network providers face challenges in operating networks, which consist of a vast number of network devices, for example, switches, routers, and gateways. IT departments must configure thousands of devices in order to implement network-wide services, which will result in difficulty to maintain Quality of Service (QoS), security and other policies. Current networks are built based on the monolithic or vertical approach, which integrates the control and forwarding functionality in a box, with vendor system standardization being a secondary consideration. The different vendor designs and construction approaches limit interoperability and have a detrimental effect on flexible network service innovation.

Global digital networks are evolving to cope with increasing traffic volumes and connected devices. There is a need for next generation network management and control systems that provide flexibility and device programmability, to facilitate dynamic updates and the introduction of new network services without hardware replacement. SDN offers one approach in providing “programmable networks” and vendors have generally adopted SDN as the next evolution of computer networking. SDN decouples the control plane from the forwarding plane in network devices and carries out traffic management utilizing a hierarchy of systems known as controllers (Foundation, 2012). Controllers connect to network switching and routing devices using open interfaces and protocols, e.g. OpenFlow (McKeown et al., 2008).

Scalability of the network has been one of the active and contentious topics in SDN. There are two common approaches in SDN controller implementation to improve the scalability, which includes centralised and distributed approaches. A centralised model is the simplest one, and it relies on the increase performance of standard controller. However, it introduces a single point of failure (SPOF) to the network. Distributed controller model eliminates SPOF and improves the scalability of the network, but it needs a method to coordinate all the controllers which could be in different domain.

A domain in SDN can be referred to an SDN administrative domain. Multi-domain SDN requires interconnection of controllers in different domains to exchange information across domain. Multi-domain SDN will enable the interconnection of global SDN domains, introduce interoperability between domains, and provide better provisioning of cross domain services. Currently, some ongoing researches are being done in multi-domain SDN, such as its architectures (Helebrandt & Kotuliak, 2014), distributed multi-domain controller architecture (Phemius et al., 2014), inter-domain communication platform (Lin et al., 2015) and application (Jahan et al., 2014), and routing mechanism (Kotronis, Gämperli, & Dimitropoulos, 2015).

Several surveys have studied SDN from different points of view. Jarraya, Madi, and Debbabi (2014) had compiled a survey on SDN providing the first taxonomy to classify SDN research works. Nunes, Nguyen, Turletti, Mendonca, and Obraczka (2014) studied the state of the art of programmable network with the emphasis on SDN, along with its implementation alternatives, and its promising research directions. Another survey by Farhady, Lee, and Nakao (2015) successfully present deep understanding of all three SDN layers. Recently, survey paper by Kreutz et al. (2015) presented a comprehensive survey on SDN which covered almost all aspect of SDN, starting from its definition, architecture and applications, until the current ongoing research efforts and challenges. However, those surveys do not present or only mentioned at very high level the multi-domain implementation aspects of SDN. Therefore, this article focuses on the multi-domain aspect of SDN.

In this paper, our aim is to provide an overview of the recent developments in multi-domain SDN (using academic and industry sources), and analyse the main research issues and approaches for future multi-domain SDN developments. The key contributions of this paper are:

- a tutorial on SDN and OpenFlow that includes a discussion of their origins, architecture and principal components.
- a review of controller implementation, both open-source and commercial, and a table with a comparison of the controller features.
- a review of current research into multi-domain SDN and the major challenges to be addressed by future research.

The rest of this paper is organized as follows. In section 2, we present a brief overview of SDN, including its architecture and the OpenFlow protocol. Section 3 discuss the multi-domain implementation of SDN. in section 4, we present multi-domain SDN challenges and identify the future research in multi-domain SDN. Section 5 concludes the paper.

2. Overview Software Defined Networking

Existing networks were generally built with proprietary hardware and systems from a single vendor. This would lead to a vendor "lock in", where it was difficult to shift to another vendor or to adopt multi-vendor solutions. The use of vendor specific network devices and systems often led to the organisation's systems becoming tailored to match the intricacies of the vendor equipment and systems as illustrated in Fig. 1. Programmable networks, that permitted the separation of the control and data planes, were seen to be the solution to the vendor "lock in" problem and the introduction of low-cost white label SDN-enabled networking devices provided a more flexible approach that organizations are now beginning to exploit. Although programmable networking was first introduced in the late 1990s, SDN has revolutionized the shift to programmable networking and SDN has become the focus of next generation networks.

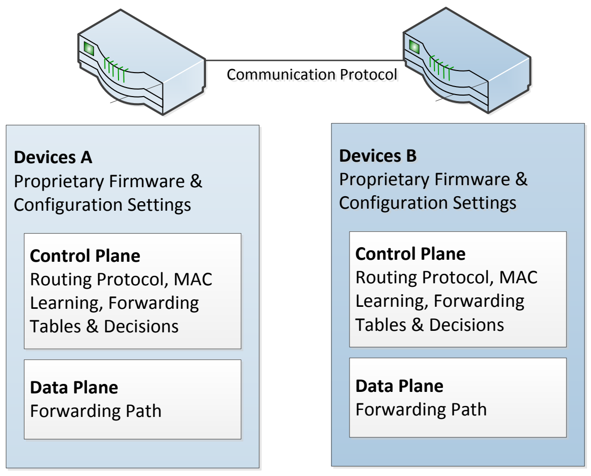

Communication Protocol
Devices A
Proprietary Firmware & Configuration Settings
Control Plane
Routing Protocol, MAC Learning, Forwarding Tables & Decisions
Data Plane
Forwarding Path
Devices B
Proprietary Firmware & Configuration Settings
Control Plane
Routing Protocol, MAC Learning, Forwarding Tables & Decisions
Data Plane
Forwarding Path

Fig. 1 Proprietary Network Element Configuration

2.1. SDN Background

SDN has evolved over the past decade to provide a more flexible and dynamic networking architecture that incorporates improved support for management and network services. The SDN approach is for the management of traffic flows to be decoupled from the underlying infrastructure and systems that forward traffic. A standardized and open protocol was introduced to facilitate the separation of control and data planes. This protocol, known as OpenFlow, was developed to facilitate control traffic transfer between management systems, known as controllers, and the network devices, such as a switch, that forward data traffic.

The development of SDN originated from the early work on programmable networking and the separation of control logic from the data transfer mechanism. There are two schools of thought regarding the concept of programmable networking including active networks and open signaling. However, the idea of decoupling the control logic from the data transfer mechanism emerged later, as a new architecture, to reduce the complexity of the distributed computations.

2.1.1. Programmable Networks: Active Network and OPENSIG

The active network concept was introduced in the mid-1990s in an endeavor to control a network in real-time. Active networking introduced a method that permits packets flowing through the network to carry instructions to be executed at network nodes. The code carried within the packets alters the network operation either temporarily for an individual packet or for a stream of packets. In this approach, the network devices become a dynamically programmable environment that can be dynamically altered using the code carried by the packets, which differs from the rigidity of traditional networking (Farhady et al., 2015; Xia, Wen, Foh, Niyato, & Xie, 2015).

Implementations of active networks include SwitchWare (Alexander et al., 1998) and conventional computer routing suites such as Click, XORP, Quagga, and BIRD (Xia et al., 2015). With the active networking implementations, the operations and behavior of the network can be modified dynamically. Although the active networking approach offered a new paradigm by providing a more dynamic environment, there was only minor development of the control plane. The active networking approach placed the intelligence at the end points (which can be inferred to be computers and servers acting as smart devices) whilst utilizing enhanced switches and routers to execute and carry out limited tasks based on the instructions carried within packets traversing the network. Thus, in active networking, packets are entities that can determine or control how nodes manage packets and streams.

In addition to the active networking approach introduced by the IP network community, another method known as Open Signalling (OPENSIG) was proposed by the telecommunication network community (Campbell, Katzela, Miki, & Vicente, 1999). The OPENSIG suggested to provide an access to network hardware by means of open and programmable network interfaces. This idea was motivated by the need to separate the communication hardware and control software. It was a thought-provoking idea due to the vertically integrated architecture of network devices (e.g. routers and switches). OPENSIG suggested that the well-defined programmable network interfaces will lead to a distributed programming environment which provides open access to switches and routers and facilitates the entry into the telecommunication software market of third-party software providers.

2.1.2. Separation of Data and Control Plane

In parallel with the programmable networking effort, innovations have emerged on the separation of the control and data planes. The aim of the innovations was to develop open and standardized interfaces along with a logically centralized network control model. The innovations were developed by a project called ForCES (Haleplidis et al., 2015) conducted by Internet Engineering Task Force (IETF). This project provided improvements to networking with the use of distributed packet processing network elements. Other approaches to developing a centralized network control model were proposed including Routing Control Platform (RCP), Path Computation Element (PCE) and Intelligent Route Service Control Protocol (IRSCP). These proposals were focused on improving network operation and the need for coexistence and backward compatibility prevented them from being deployed immediately (Jarraya et al., 2014).

Other initiative, commenced in the period of 1990, included the Devolved Control of Asynchronous Transfer Mode (ATM) Networks (DCAN), which emerged from the scalable control and management objective for ATM networks. This initiative suggested the removal the control and management functions from ATM based network devices. Those functions should be assigned to external entities, dedicated to that function. DCAN also assumed a simple protocol between the manager and the network, which aligns with today's OpenFlow protocol (Nunes et al., 2014).

In 2004, the 4D project (Greenberg et al., 2005) introduced a clean slate design, which stressed the separation of the routing decision logic and the protocols used to pass messages between network elements. The proposal gave a global view of the network with a decision plane, serviced by a dissemination and discovery plane used to control the data traffic forwarding. Later, these ideas provided the inspiration for advanced research into NOX, the first SDN controller, which in the context of an OpenFlow-enabled network is also known as an 'operating system for networks'.

2.1.3. Other Pre-SDN Projects

The IETF Network Configuration Working Group proposed NETCONF in 2006, as a management protocol for network device configuration. The protocol carried messages to network devices through an exposed API that supported extensible configuration data. NETCONF had efficiency, effectiveness, and security advantages when compared to the Simple Network Management Protocol (SNMP), a popular network management protocol, especially when managing complex and diverse network environments (J. Yu & Al Ajarmeh, 2010). SNMP and NETCONF are both useful management tools that can be used in parallel on hybrid switches that support programmable networking.

Other protocols that preceded OpenFlow were the SANE (Casado et al., 2006) and Ethane projects (Casado et al., 2007) completed in 2006. They defined a new architecture for enterprise networks that utilized a centralized controller to manage policy and security. Identity-based access control was its significant feature. Like SDN, Ethane used two components: a controller and Ethane switch. The controller decides whether a packet should be forwarded, while the switch connects to the controller via a secure channel and holds a flow table providing the foundation for SDN.

Table 1
SDN Standardization Activities

Standardization Organizations	Scope of Work
ONF	SDN Architectures and its components, SDN interfaces, OpenFlow protocol extensions, OpenFlow Switch Specifications, OpenFlow Configuration and Management Protocol,
IETF	ForCES Protocol, SDN Architechture, OpenFlow interworking, Control Plane Requirements,
ITU-T	Signalling requirements using SDN technologies in Broadband Access Network, Functional architecture for SDN, SDN Control of Transport Network, and Security aspect in SDN
Broadband Forum	Requirements and impacts of deploying SDN in Broadband Networks
IEEE	Applicability of SDN to IEEE 802 infrastructure
IRTF	Prospection of SDN for the evolution of Internet
MEF	Service orchestration in Network as a Service

2.2. SDN Standardization Efforts

To date, several standardization organizations have focused on SDN and some SDN-related standards have been published, as shown in Table 1. There is ongoing work to define further SDN and associated technologies (Farhady et al., 2015; Hakiri, Gokhale, Berthou, Schmidt, & Gayraud, 2014; Kreutz et al., 2015; Nunes et al., 2014; Xia et al., 2015).

The Open Network Forum (ONF) released documentation on the current SDN Architecture, focusing on the OpenFlow protocol. The IETF with its Forwarding and Control Element Separation (ForCES) Working Group has focused on standardizing mechanisms, interfaces, and protocols targeting the centralization of network control and abstraction. The International Telecommunication Union-Telecommunication Sector (ITU-T) has set up several Study Groups (SGs) which have developed SDN recommendations. A Joint Coordination Activity on SDN (JCA-SDN) was formed to coordinate the SDN standardization effort between groups.

The Broadband Forum (BF) considered SDN and Network Function Virtualization (NFV) to be within the Technical Work Area of its Technical Committee Work in Progress and is currently investigating the migration and deployment of SDN and NFV-enabled implementations across all aspects of the broadband network. At the IEEE, the 802 LAN/MAN Standards Committee has recently initiated several activities to standardize SDN for access networks based on the IEEE 802 infrastructure through the P802.1CF project, for both wired and wireless technologies to embrace new control interfaces. The Metro Ethernet Forum (MEF)'s aim in SDN was to define the service orchestration with APIs for existing networks. The Internet Research Task Force (IRTF) created the Software Defined Networking Research Group (SDNRG) to investigate SDN from its perspective with the goal of identifying alternate approaches and future research opportunities.

The open source software community continues to work on controllers and networking stacks including OpenDaylight, OpenStack, and CloudStack. This effort aims to develop the basic building blocks to support SDN and NFV

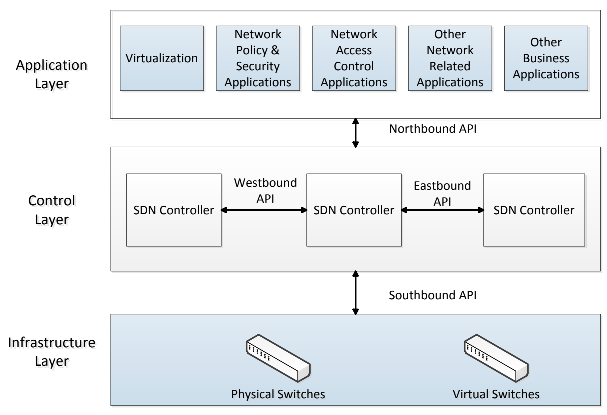


Fig. 2 SDN Architecture.

implementations. For example, OpenDaylight is intended to be extensible and configurable to support potentially emerging SDN open standards e.g. OpenFlow, I2RS, VxLAN, PCEP.

2.3. SDN Architecture

The ONF defined SDN as an emerging network architecture where network control is decoupled from forwarding and is directly programmable (Foundation, 2012). The control function is migrated from formerly tightly bound devices in each network to accessible computing devices. This migration enables the underlying infrastructure to be abstracted for applications and network services. Later, the network could be treated as a logical or virtual entity.

There are two major SDN characteristics, as depicted in Fig. 2, including decoupling of the control and data planes, and control plane programmability. Both have previously been the focus of extensive research and recent improvements in the reliability, capacity and capability of global networks that have enabled the control plane programmability concept to move forward. SDN encompasses the separation of control and data planes in the network's architectural design, which means that network control is to be carried out utilizing separate channels between device control management ports that utilize different addresses to that used for the data plane. The network intelligence is taken out of the switching devices, thereby leaving the switching devices as general forwarding devices.

There are three functional layers in SDN architecture, i.e. Application Layer, Control Layer, and Infrastructure Layer. Each layer had its own functions and communicate to each other via an specific interface (Hakiri et al., 2014; Jarraya et al., 2014). The descriptions of SDN layers are presented below:

a) Application Layer

   The Application Layer consists of network services and applications that can be abstracted using the dynamic modular structure of the Application Layer. Examples of network services and applications include management systems, monitoring, security and flow control related network services.

   The network abstraction utilizes an Application Programming Interface (API) to provide consistency and standardization of the interface. Through this API, SDN services and applications can access network status information reported from forwarding devices. SDN services and applications can also use this API to transfer flow rules to forwarding devices through the lower layers.

b) Control Layer

   The Control Layer consists of a set of software-based SDN controllers that provide control functionality through open API-based interfaces that facilitate the control and management of traffic forwarding. The controllers incorporate three communication interfaces, i.e. southbound, northbound and eastbound/westbound.

   The control plane acts as an intermediary layer between the application and data planes. The controller provides a programmatic interface that can be accessed by network services and applications in the Application Layer and used to implement management and control tasks. The abstraction presumes the centralized control and the applications are developed within the framework of a single interconnected system. It enables the SDN model to be implemented for a broad range of scenarios, such as centralized, hybrid or distributed and also for heterogeneous network technologies (wireless or wired). The controller implementation design significantly affects the overall performance of the network. Several challenges must be overcome to achieve network performance that is at or above the network performance of the preceding legacy network.

c) Infrastructure Layer

   The Infrastructure Layer is the lowest layer in the SDN architecture. Forwarding elements are the main components in this layer, which include physical and virtual routers and switches. These devices are accessible via an open interface and carry out packet routing, switching and forwarding. The control connections to the network devices utilize separate secure channels to that used for user data flows.

SDN architecture employs a three specific interfaces. These interfaces enable the interactions between and within SDN layers. Below, we present the description of SDN interface:

a) Northbound Interface

   The Northbound interface is used to connect network services and applications found in the Application Layer to the controllers in the Control Layer. The Northbound interface consists of one or more API providing a programmability capability that is used to dynamically manage network traffic flows. It is more considered as a software API rather than a protocol based interface, to take advantage of the innovative programmability paradigm. The ONF suggest a definition that the different levels of abstractions are latitudes and the various use cases are longitudes, but this characterisation is yet to be finalized. The ONF approach suggest that more than a single northbound interface standard can provide increased flexibility to serve differing use cases and environments. Representational State Transfer (REST) is one of the proposed APIs to provide a programmable interface for business applications to the controllers (Jarraya et al., 2014).

b) Eastbound/Westbound Interface

   The Eastbound/Westbound interface is a proposed communication protocol related interface, which is yet to be fully standardized. It is identified to enable communication between groups or federations of controllers to synchronize states for high reliability and resiliency. The interfaces are to be used to cope with the SDN scalability and reliability challenge. The Eastbound/Westbound interface protocol manages communications between the multiple controllers. There are two possible use cases for this interface. The first use case is an interconnecting interface between conventional IP networks with SDN networks. As there are no standards defined for this interface, its implementation depends on the technology used by the underlying network. An example of this use case is the connection between SDN domain with a legacy domain using a legacy routing protocol to react to message requests (e.g., Path Computation Element (PCE) protocol and MPLS). The second use case is to use the interface as an information conduit for admission and authentication, between the SDN control planes of different SDN domains. The multi-domain connectivity challenge needs to be overcome to facilitate a global network view and influence the routing decisions of controllers on domain boundaries. A solution would allow a seamless setup of network flows across heterogeneous SDN domains. Conventional border protocols, like BGP, could be utilized or extended to support interconnection of remote SDN domains.

c) Southbound Interface
This interfaces facilitate the communication between SDN controller with the Forwarding Elements in the infrastructre layer. The standardized protocol for this communication is OpenFlow. It is developed and maintained by ONF. OpenFlow is described as the fundamental element of SDN solution development. OpenFlow allows multi-vendor SDN network devices to be implemented. Other alternarives to OpenFlow is the Forwarding and Control Element Separation (ForCES) Framework (Haleplidis et al., 2015). The latter defines an architectural framework with coupled protocols to standardize information exchange between the control and forwarding layers. Although it has been an IETF proposal for several years, it has never reached the level of support that exists for OpenFlow.

2.4. OpenFlow as SDN Enabler

OpenFlow is an open source communications protocol that is used to transport messages from controllers to network devices via the Southbound interface. OpenFlow provides software-based access to the switch and router flow tables to enable dynamic network traffic management. Manual or automated control systems can be used depending on the specific network management operations scenario. The OpenFlow protocol provides a management tools to manage features such as topology configuration or packet filtering.

OpenFlow shares common ground with the architectures proposed by ForCES and SoftRouter. The difference is in the concept of flow management and leveraging the existence of flow tables in commercial switches and routers (Braun & Menth, 2014).

Switches with OpenFlow compliance are categorized in two main types, i.e. OpenFlow-only switch and OpenFlow-hybrid. OpenFlow-only switches support only OpenFlow operations; i.e., all packets are processed and controlled using the OpenFlow pipeline to the upstream controller. OpenFlow-hybrid switches support both OpenFlow

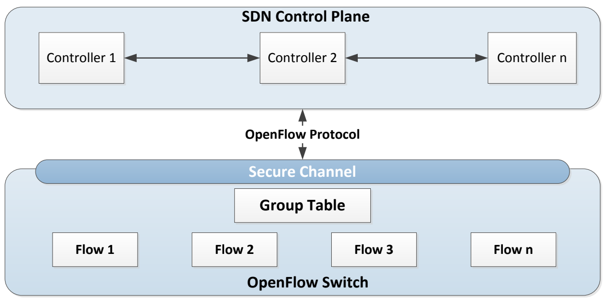


Fig. 3 OpenFlow Elements

operations and legacy Ethernet switching operations. The hybrid switches support a classification mechanism outside of OpenFlow that routes traffic to either of the packet-processing pipelines (Farhady et al., 2015; Jammal, Singh, Shami, Asal, & Li, 2014; McKeown et al., 2008).

The OpenFlow architecture consists of numerous OpenFlow-enabled switches that are managed by one or more OpenFlow controllers, as shown in Fig. 3. Network traffic can be partitioned into flows, where a flow could be a Transmission Control Protocol (TCP) connection, packets with the same MAC address or IP address, packets with the same Virtual Local Area Network (VLAN) tag, or packets arriving from the same switch port. Several elements of the OpenFlow architecture are explained in this section.

2.4.1. OpenFlow Protocol

An OpenFlow-enabled switch contains flow and group tables that include a number of entries, depending on the network device. The flow entries are used to control traffic flows arriving at the switch using identifiers such as source, destination and IP port. OpenFlow messages manipulated the flow entries in the flow table. The messages are exchanged between the switch and the controller via a secure channel. By maintaining a flow table, the switch can make forwarding decisions for incoming packets using a simple look-up on the entries of its flow table. The switch will do an exact match check on particular fields of the incoming packets. The switch goes through its flow table to find a matching entry for incoming packets. Numbering in the flow tables are done sequentially, starting from 0. The packet processing pipeline starts at the first flow table and if a match is not found it moves on to the next flow table and so on until a match is found, or the end of the last flow tables is reached. If the specific packet fields match a flow entry, the corresponding instruction set is executed. Instructions related to each flow entry describe packet forwarding, packet modification, group table processing, and pipeline processing.

Pipeline-processing instructions enable the packet fields to be matched to a flow entry in one table and based on the flow entry instructions be sent to associated tables for additional matching and processing, which can result in an aggregation of actions that are to occur to the packet before transmission. The aggregated information (metadata) can be communicated between flow tables. Flow entries may also forward to a physical or virtual port.

Flow entries may link to a group table entry, which specifies additional processing. Additional forwarding methods (multicast, broadcast, fast reroute, link aggregation) are offered by a group entries inside a group table. A group entry contains a group identifier, a group type, counters, and a list of action buckets, which contain a set of actions to be executed and associated parameters. Groups also allow multiple flows to be forwarded to a single identifier, e.g., IP forwarding to a common next hop. Occasionally, a table miss occurs, where a packet might not match a flow entry in any of the flow tables. The action taken in the case of a miss depends on the table configuration. By default, selected packet fields are sent to the controller over the secure channel. Another option is to drop the packet.

OpenFlow was first released by Stanford University in 2008 (McKeown et al., 2008). Since 2011, the OpenFlow switch specification has been maintained and updated by the ONF. OpenFlow vendors have widely adopted the latter version. In the initial ONF version, forwarding was based on a single flow table, and packet matching using Layer 2 information and IPv4 addresses. Version 1.1 introduced the multiple flow tables and MPLS tags, while IPv6 support was included in version 1.2. Version 1.3 added the support for multiple parallel channels between switches and controllers. In version 1.4, improvements include retrofitting various parts of the protocol with the Type/Length/Value (TLV) structures introduced in version 1.2 for extensible field matching and a flow monitoring framework enabling a controller to monitor in real-time the changes made to flow tables within other controllers. The latest OpenFlow specification published in 2014 is version 1.5, (with minor improvements in 2015 becoming version 1.5.1) which introduces new features such as an Egress Table, Packet type aware pipeline, and a Flow Entry Status Trigger.

2.4.2. OpenFlow Switch

An OpenFlow-enabled switch contains at least one flow table and group table. It performs packet lookups and forwarding. Controller manages the switch utilizing the OpenFlow protocol via a secure channel. A set of flow entries makes a flow table in the switch. Each flow entry is used to match to the packet header fields, counters, and a set of instructions for matching packets (Jammal et al., 2014).

2.4.3. OpenFlow Channel

The OpenFlow channel is the interface that connects OpenFlow-enabled switches to a controller. The controller configures and manages the switch, using this interface. The OpenFlow protocol supports three message types that transport the control messages across the secure channel. The messages can be categorized as controller-to-switch, asynchronous, and symmetric, each having multiple subtypes. Controller-to-switch messages are used to manage or derive information directly regarding the switch state. These messages are initiated by the controller. Asynchronous messages are initiated by the switch. These messages are used to update the controller with changes to the switch state and network events. Symmetric messages are initiated by either the switch or the controller and are sent without solicitation. The OpenFlow channel is usually encrypted using transport layer security (TLS), but can also operate directly over TCP (Jammal et al., 2014).

2.4.4. OpenFlow Controller

The controller is responsible for maintaining the network protocols, policies and distributing appropriate instructions to the network devices. In other words, the OpenFlow-enabled controller is responsible for determining how to handle those packets that do not match valid flow entries found in the switch flow tables. It manages the switch flow table by adding, modifying and removing flow entries over the secure channel using the OpenFlow protocol. The switch must be able to communicate with a controller at a user-configurable (but otherwise fixed) IP address using a user-specified port and recent implementations provide uPnP to enhance connectivity options. A standard TLS or TCP connection to the controller is started by the switch, based on controller's IP address. Traffic coming in and out the OpenFlow channel does not travel through the OpenFlow pipeline. Hence, the switch must identify incoming traffic as local before matching it compared to the flow tables. A single or multiple controllers can establish communication with the switch (Jammal et al., 2014).

Reliability could be improved by having multiple controllers, because the switch can continue to operate in OpenFlow mode if one controller connection fails. The controllers themselves managed the hand-over between controller, which enables load balancing and fast recovery from failure. The controllers coordinate switch management based on rules set by higher layer applications, and the goal of the multiple controller functionalities is to synchronize controller hand-offs performed by the controllers.

2.4.5. Flow and Group Tables

There are three fields in flow table entries, i.e. Packet Header, Action, and Statistic. Packet header which is unique to the flow, defines the flow, and is almost a ten-tuple. Its fields contain information such as VLAN ID, source and destination ports, IP address, and Ethernet source and destination. Action field specifies how to handle the packets in the flow. An example is for the switch to forward the packet to a given port or ports, drop the packet, or forward the packet to the controller. Statistics field includes information such as the number of packets, the number of bytes, and the time from when the last packet matched the flow for each type of flow. Counters are typically used to monitor the number of packets and bytes for each flow, and the elapsed time since flow initiation (Jammal et al., 2014).

Besides Flow table, an OpenFlow switch also has Group table, which consists of group entries. A flow entry can point to a group, which characterise OpenFlow's new method of forwarding (e.g. select and all). Each Group table entry has four fields. The first field is a Group Identifier, which is a 32-unsigned integer that uniquely identified the group. The second field is group type, which determines group semantics. The third field is Counters, which is updated when packets are processed by a group. The last field is Action Buckets, which is an ordered list of action buckets. Each action bucket contains a set of actions to execute and associated parameters (Jammal et al., 2014).

3. Multi-Domain in Software Defined Networking

SDN offers flexibility in managing the flows inside a network. In the other hand, global digital network is increasing on its size and complexity. Each network domains are connected to each other to become a massive and large network. With its advantages, SDN could provide more efficient way in managing the flows and network.

Multi-domain SDN emerges as one of the solutions in implementing SDN in large network. The need to enhance current networks operations and managements motivates the implementation of SDN in them. Nevertheless, SDN hasn't yet equipped to operate in very large network, e.g. there hasn't any standard yet for East/Westbound interfaces. Therefore, there are a great deal of works need to be done in this area.

In this section, we discuss the works on multi-domain SDN, starting from the challenges in implementing SDN in SDN in large network from the controller view. Some approaches to overcome the scalability issues and lists of supporting SDN controllers are also presented. At the end of the section, we will discuss the multi-domain SDN architectures.

3.1. Challenges in SDN Controller Design

Controller design requires substantial effort if the controller is to provide flexible interfaces for network services and applications and a verified OpenFlow interface. It is more than just a matter of designing and implementing the interfaces, matching the interfaces with the network and applications, programming languages, and software architecture in the controller; the design also relates to the performance of SDN-enabled networks.

3.1.1. Scalability

An initial concern that arises when offloading control from the switching hardware is the scalability and performance of the network controller(s). SDN's centralized control methodology naturally faces scalability issues. The controller is the most important artifact in the SDN architecture and a single controller elucidation may result in a single point of failure and performance bottleneck problems in a wide area SDN (Farhady et al., 2015; Nunes et al., 2014). The entire network will break down if the controller fails. On the other hand, no matter where we place the controller, it will be farther away from some of the switches under its control. These switches will experience higher flow setup latency (Karakus & Durresi, 2016). Clearly, a single controller solution is not suitable for wide-area SDN and some enterprise networks.

Other concerns on the scalability of SDN in large network are the aggregating and disseminating a huge number of information, both from and throughout the network (Shuhao & Baochun, 2015). Those processes need to be done in real-time, which make things worse. The proposed solutions from control plane design to cope with scalability issues can be classified in two categories. The first category is using a single-controller approach. In this category, improvements are done by reducing the overhead of the centralized controller in several aspects (Farhady et al., 2015; Karakus & Durresi, 2016).

The second category is using multi-controller approach. For a multi-controller SDN wide area solution, two alternatives are possible: replicated and distributed (Kreutz et al., 2015). Multiple replicated controllers can improve fault resilience. A replicated approach is to maintain an online shadow controller that will take over only if the primary controller fails. Switches are configured to communicate with both controllers simultaneously as the alternative approach, which is switch replication, remains to be implemented (Shuhao & Baochun, 2015). Using a shadow switch may produce a significant communication overhead in a WAN, where the controller and switches could be several hops apart. For short flows, this might generate more flow setup traffic

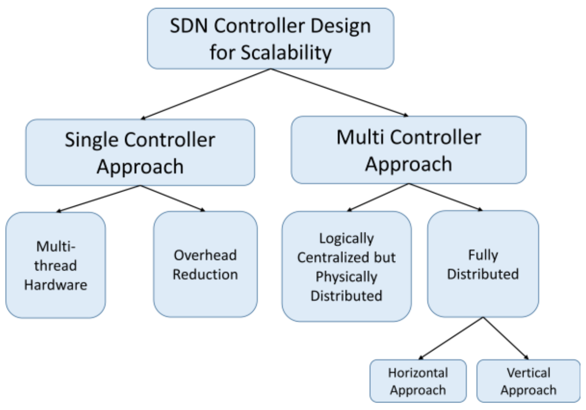

SDN Controller Design for Scalability
Single Controller Approach
Multi Controller Approach
Multi-thread Hardware
Overhead Reduction
Logically Centralized but Physically Distributed
Fully Distributed
Horizontal Approach
Vertical Approach

Fig. 4 SDN Control Design for Scalability

than the flow itself. In distributed controller architecture, controllers are responsible for a portion of the network. Therefore, distributed controller solution should yield improved performance and robustness than replicated solution (Ahmed & Boutaba, 2014).

3.1.2. Placement and Reliability

In the wide area SDN implementation, controller(s) location may impact on the overall network performance. Whether the SDN consists of single or multiple controllers, the placement of the controller(s) will have an impact on the performance and the cost of the network. Research is ongoing into architecture, location and the number of controllers with respect to the average and worst case latencies of control plane (Heller, Sherwood, & McKeown, 2012) and other metrics, e.g. latency in case of failure and inter-controller latency (Vizarreta, Machuca, & Kellerer, 2016)

The research shows that latency drives the overall behavior of the network, and bandwidth for the control traffic affects the number of flows that the controller can process. The modeling of the network is used to identify controller locations that enhance reliability and limit control message latency (Sallahi & St-Hilaire, 2015). While other alternative approach tries to solve the controller(s) placement issue, focuses on optimizing the reliability of the control network and identifies several placement algorithms and strategies along with metrics to characterize the reliability of SDN control networks (Jimenez, Cervello-Pastor, & Garcia, 2014; Vizarreta et al., 2016).

3.1.3. Security

SDN controllers may suffer from a range of security problems, which can reduce the performance of the network. The attacks might affect performance due to the lack of controller scalability in the event of a denial of service attack. The impact could be worst in the large network with only a single controller or even for the multiple controllers.

Those attacks can aim the forwarding layer, control layer, and application layer, and their types can be discussed as follow (Dabbagh, Hamdaoui, Guizani, & Rayes, 2015; Kaur, Singh, Singh, & Sharma, 2016). On the application layer, the attacks could be trough unauthenticated application and policy enforcements. Denial of Service (DoS) are the attack that target the forwarding and control layer. DoS could be caused by the massive flows that flood the switches and controllers which cause processing delay or device collapsed.

Other type of attacks is the compromised controller attack, which happens when the attacker somehow gain access to the controller and could take over all the network. Data leakage and flow rule modification are the impact of attacks on forwarding layer input buffers. is often disastrous. Controller hijacking and fraudulent rule Insertion attacks are types of malicious applications. DOS (Denial of services) are related to availability related attacks. Type of DOS attack is Controller-Switch Communication Flood between the switch and the main controller.

There are some possible countermeasures for those attacks. Attacks targeted the forwarding plane could be avoided by proactive rule caching, rule aggregation, increasing switch's buffering capacity, decreasing switch-controller communication delay, and packet type classification based on traffic analysis. Other attacks that targeted control and application plane could be mitigated by controller replication, dynamic master controller assignment, efficient controller placement, controller replication with diversity, and efficient controller assignments (Dabbagh et al., 2015).

3.1.4. *Availability*

SDN in large network could suffer from various failures Those failures including controller/switch failure and link failure. SDN controller can be overloaded due enormous request from networks. A SDN-enabled switch could confront with a failure if any of its sub-component does not function correctly. This causes the traffic routing/forwarding functions of the network not constantly retained. While link failures can occur on a network link as device connection is broken. The cause of this failure could be from network connectivity and hardware problems, or software problem in any network device that generate link down notifications falsely (Nguyen et al., 2015).

In SDN networks, the overall design, including placement and selection of network devices such as the controllers and switches, should be robust, and this should be tested for a range of anticipated scenarios. An approach that improves the robustness of SDN is to use a runtime system that automates failure recovery by spawning a new controller instance and replaying inputs observed by the old controller. The controller can install static rules on the switches to verify topology connectivity and locate link failures based on those rules. Another approach is to try to improve recovery time by the frequent issuing and receipt of monitoring messages, but this may place a significant load on the control plane (Farhady et al., 2015). In multi-controllers SDN, a load balancing mechanism based on a load informing strategy is proposed to dynamically balance the load among controllers (Jinke, Ying, Keke, Shujuan, & Jiacong, 2016).

3.2. Wide Area SDN Control Plane Design

SDN controller designs for large or wide area network mainly aimed to solve the scalability issue. As presented in **Error! Reference source not found.**, the approaches proposed to overcome this challenge can be categorized in two big categories, i.e. single controller solution and multiple controller solution (Xia et al., 2015). The single controller approach implements multi-thread hardware and the overhead reduction of the centralized controller in certain scenarios. The multiple controller approach consists of logically centralized and fully distributed or multi-domain approaches (Xia et al., 2015). In this section, we will discuss both approaches and the summary of supporting controller platforms.

3.2.1. *List of Available of Controller*

Currently, many SDN controllers are available, both open source and commercial ones. Those controllers have their own specific features and support, especially in wide area implementation. A summary of those controller is presented in Table 2.

3.2.2. *Single Controller Approach*

SDN may have either a centralized or distributed control plane (Hakiri et al., 2014; Kreutz et al., 2015), although the protocols such as OpenFlow specify that a controller controls a switch, and this seems to imply centralization. OpenFlow is not defined for controller-to-controller communication, but it is apparent that something similar is needed for distribution or redundancy in the control plane.

The single controller approach, shown in Fig. 5, is based on a single centralized controller that manages and supervises the entire network, which is already supported by the ONF. In this model, network intelligence and states are logically centralized inside a single decision point. This centralized controller uses the southbound protocol (e.g. OpenFlow) to conduct global management and control operations. The centralized controller must have a global view of the entire network, including the load on each switch along the routing path. It also has to monitor link bottlenecks between the remote SDN nodes. Additionally, statistical information, errors and faults from each network device can be collected by the controller from the attached switches and this information is passed on to another entity, which is often a database and analytic system that identifies switch and network loads and predicts future loads.

As mentioned before, single controller approach exploits two methods, i.e. utilizing the hardware and reducing the overhead to minimize controller loads. The first method provides performance scalability at times of high load or when a controller failure occurs. Conventional software optimization techniques can be used to improve the controller's performance. Multi-core hardware that supports multi-threading can be used to support parallel process optimization, load balancing, and replication. High-performance controllers, such as McNettle (Voellmy & Wang, 2012), target powerful multi-core servers and are

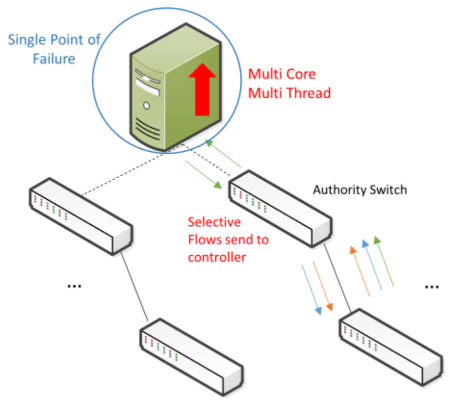


Fig. 5 Single Controller Approach

being designed to scale up to handle large data center workloads (around 20 million flow requests per second and up to 5000 switches).

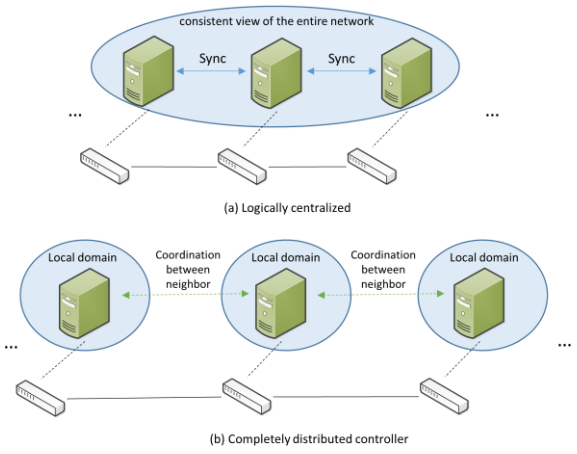
Despite the performance improvements presented in this approach, there are limitations and challenges. The hardware setup cannot be easily modified due its rigid nature. Another limitation is that this approach retains a Single Point of Failure due its single-controller architecture

The second method tries to reduce the controller load, as highlighted by proposals including as DIFANE (M. Yu, Rexford, Freedman, & Wang, 2010) and DevoFlow (Curtis et al., 2011), by extending the switch data plane mechanism. DIFANE partly uses intermediate switches called authority switches to make the forwarding decisions instead of totally relying on the centralized controller to reduce the load and the latencies of rule installation. A similar approach is also proposed in DevoFlow with the selection of particular flows to be directed to the controller, while switches handle the other flows.

3.2.3. *Multiple Controller Approach*

Multiple controllers approach, as shown in Fig. 6, offers solutions to solve the SDN scalability issues. This approach uses multiple controllers that manage and supervise the network. These controllers are distributed along the network and can be called distributed controllers. There are two classes of distributed SDN controllers, i.e. logically centralized but physically distributed controller, and the completely distributed controller.

Distributed controllers have several key challenges that should be addressed to improve the scalability and robustness of networks. The first challenge is the requirement of a consistent network-wide view in all controllers. Static configuration could not be used to overcome this challenge, because it can cause uneven load distribution between controllers. The mapping between control planes and forwarding planes must be automated. The second challenge is to gain an optimal global view of the whole network. Not to mention, identification of an optimal number of distributed controller that guarantee the linear scale up of SDN network, is another hard effort. The last challenge is how synchronize the overall local and distributed events to provide a global picture of the network. Most of the approaches use local algorithms to develop coordination protocols, in which each controller needs to respond only to events that take place in its local neighbourhood.


consistent view of the entire network
Sync
Sync
...
...
(a) Logically centralized
Local domain
Coordination between neighbor
Local domain
Coordination between neighbor
Local domain
...
...
(b) Completely distributed controller
Fig. 6 Multiple Controller Approach

The first class of distributed controllers is the logically centralized but physically distributed controller. These controllers have to share information with each other so as to build a consistent view of the entire network. They are using either distributed file system, a distributed hash table, or a pre-computation of all possible combination to centralized their logic. This approaches impose a strong requirement: a strongly consistent network-wide view in all the controllers. The network-wide view is maintained via controller-to-controller synchronization. When the local view of a controller changes, the controller will synchronize the updated state information with the other controllers. The information exchange or state synchronization among controllers consumes network resources; therefore, it is critical to reduce the resulting network load, while keeping the information consistent for the logically-centralized control plane. The example of this implementation are:

a. Onix (Koponen et al., 2010) focuses on the problem by providing generic distributed state management APIs. With Onix, the control plane operates with a global view of the network.
b. Hyperflow (Tootoonchian & Ganjali, 2010) suggests a logically centralized control which consists of many distributed controllers and exhibits excellent scalability. It has been implemented as an application for a NOX. In reality, network operators can deploy any number of controllers on demand. By propagating events that affect the controller's state, HyperFlow can enable all of the controllers to obtain a network-wide view by passively synchronizing network-wide views of OpenFlow-enabled controllers. Since each controller has a global view, HyperFlow minimizes the response time of the control plane through local decision making by each controller.
c. Google has presented their experience with B4 (Jain et al., 2013), a global SDN deployment interconnecting their data centers, using a logically centralized Traffic Engineering (TE) service to decide on path computation.
d. Devolved controllers (Tam et al., 2011) propose control plane distribution using pre-computation of other inter-relationships. Devolved controllers require a strongly consistent network-wide view to be maintained in the controllers.

Table 2
Summary of Controller Platforms

Controller Name	Prog. Language	Organization	Open Source	Architecture	Description
Beacon	Java	Stanford	Yes	Centralized Multi-threaded	Cross-platform, modular, Java based OpenFlow Controller that support event-based and threaded operation (Erickson, 2013).
Maestro	Java	Rice Univ.	Yes	Centralized Multi-threaded	A network operating system based on Java which provides interfaces for implementing modular network control applications (Cai, 2011).
FloodLight	Java	Big Switch Network	Yes	Centralized Multi-threaded	Java-based OpenFlow Controller, based on the Beacon implementation, works with physical and virtual OpenFlow switches (Floodlight).
RISE	C & Ruby	NEC	Yes	Centralized	OpenFlow Controller based on Trema, which is an OpenFlow stack based on Ruby and C (Ishii et al., 2012).
ONOS	Java	ON.Lab	Yes	Distributed	Open source SDN controller platform designed specifically for scalability and high-availability. With this design, ONOS projects itself as a network operating system, with separation of control and data planes for Wide Area Network (WAN) and service provider networks (Berde et al., 2014).
RYU	Python	NTT	Yes	Centralized Multi-threaded	SDN operating system that aims for logically centralized control and APIs, to create new network management and control applications (Kubo, Fujita, Agawa, & Suzuki, 2014).
NOX/POX	Python	Nicira	Yes	Centralized	The first OpenFlow controller (Gude et al., 2008).
OpenContrail	Python	Juniper	Yes	Distributed	An open source version of Juniper’s Contrail controller (Lin et al., 2015; Singla & Rijsman, 2013)
OpenDaylight	Java	Linux Foundation	Yes	Distributed	An open source project based on Java. It supports OSGi Framework for local controller programmability and bidirectional REST for remote programmability as Northbound APIs (Medved, Varga, Tkacik, & Gray, 2014; Phemius et al., 2014).
OpenMUL	C	KulCloud	Yes	Centralized Multi-threaded	C-based multi-threaded OpenFlow SDN controller that supports a multi-level northbound interface for attaching applications (Saikia & Malik, 2015).
Big Cloud Fabric controller	Java	Big Switch Network	No	Centralized Multi-threaded	The controller is part of Big Switch’s SDN-based Data Center solutions. It is hierarchically implemented SDN controller that capable to be implemented as a cluster of virtual machines or hardware appliances for high availability ("Big Cloud Fabric: Hyperscale Networking for All," 2014)
Brocade Vyatta Controller	Java	Brocade	No	Distributed	OpenDaylight Based Controller with additional support services from Brocade ("Brocade Vyatta Controller," 2015).
Ericsson SDN Controller	Java	Ericsson	No	Distributed	Ericsson used the OpenDaylight as the base of its controller and bind a Policy Control to drive end-user service personalization in network connectivity ("The Real-Time Cloud," 2014).
One Controller	Java	Extreme Networks	No	Distributed	Extreme Networks offers its SDN controller (OneController) in form of hardware appliance based on OpenDaylight controller and provides Extreme Network’s OpenDaylight-based API, Software Development Kit (SDK) and a developer community ("OneController™ OpenDaylight-based SDN Controller," 2015).
HP VAN SDN Controller	Java	HP	No	Distributed	OpenDaylight based controller with HP contributions in AAA, device drivers, OpenFlow and Hybrid Mode, clustering for High Availability, multi-application support including the Network Intent Composition (NIC) API, Persistence, Service Function Chaining, OpenStack integration and federation of controllers ("HP VAN SDN Controller Software,").
Programmable Network Controller	Java	IBM	No	Distributed	OpenDaylight based controller offered by IBM as part of its Data Center Solution ("IBM Programmable Network Controller V3.0," 2012).
Contrail	Python	Juniper	No	Distributed	Contrail Controller is a software controller that is designed to operate on a virtual machine (VM). It exposes a set of REST APIs for northbound interaction with cloud orchestration tools, as well as other applications ("Contrail Architecture," 2013).
Programmable Flow Controller	Java	NEC	No	Centralized	It provides a high performance, fabric-based SDN with advanced network automation, control, and flexibility, enabling full network virtualization and secure, multi-tenant networks ("Award-winning Software-defined Networking NEC ProgrammableFlow® Networking Suite," 2013).
Open SDN Controller	Java	Cisco	No	Distributed	OpenDaylight based SDN Controller with additional Cisco’s embedded applications, robust application development environment and additional OpenFlow protocol support for Cisco Multiprotocol Label Switching (MPLS) extensions ("Cisco Open SDN Controller 1.1," 2015).
Huawei IP SDN Controller	Java	Huawei	No	Distributed	OpenDaylight based SDN controller with addition of Huawei’s Open Programmability System (OPS), which implements multi-layer capability openness including network control and management ("Huawei IP SDN Controller Open and Application," 2013)

The second type of distributed controller model is the completely distributed controller. It introduces a physically distributed control plane state and logic; therefore, there is no synchronization of network states between controllers to maintain a global view. Synchronization could lead to a network overload due to the frequent network changes and it suffers from control state inconsistency which will degrade the performance of applications running on top of SDN.

For the distributed control plane, each controller handles its local domain and attempts to make improved routing decisions at each controller by using a local algorithm that can be utilized to develop efficient coordination using available protocols and messages, whereby each controller cooperates only with its neighboring controllers. Although the existing distributed algorithms may be used, there are also demands for a SDN dedicated distributed algorithm.

A multi-domain SDN architecture refers to a network architecture that connects multiple SDN domains. SDN domain refers to the administrative SDN domain, which might be a sub-network in a data center network, or a carrier or an enterprise network, or an Autonomous System (AS). Most of the distributed control plane architectures with a logically centralized approach such as Onix, Hyperflow, and Elasticon currently cannot manage inter-domain flows between SDN domains. According to (Egilmez, 2014), the fully distributed SDN controller architectures, both vertical and horizontal approaches could be utilized for multi-domain SDN communication. A summary of contributions in multi-domain SDN is presented in Table 3.

Table 3
Summary of Contributions in Multi-Domain SDN

Contributions	Descriptions
Elasticon	Proposed an IETF draft to connect SDN domains using an automated system (Yin et al., 2012).
Pratyaastha	Proposes a novel method for assigning SDN switches and partitions of SDN application state to distributed controller instances (Krishnamurthy, Chandrabose, & Gember-Jacobson, 2014)
DISCO	Proposed an open and extensible distributed SDN control plane able to deal with the distributed and heterogeneous properties of modern overlay networks and WAN (Phemius, Bouet, & Leguay, 2014)
Kandoo	Proposed a hierarchical model of distributed controllers, i.e. root controller and local controllers (Hassas Yeganeh & Ganjali, 2012)
ONOS	Propose the distributed architecture for high availability and scale-out SDN (Berde et al., 2014)
ODL SDNi	Propose an application that can connects multiple controllers in different domains (Jahan et al., 2014)
IETF SDNi	Proposed an IETF draft to connect SDN domains using an automated system (Yin et al., 2012).
West-East Bridge	Proposes an information exchange platform for inter-domain SDN peering (Lin et al., 2015),
Novel SDN Multi-Domain Architecture	Proposed multi-domain SDN architecture and defined an interconnection protocol (Helebrandt & Kotuliak, 2014).

Different approaches to the distributed SDN controllers are proposed by Elasticon, Pratyaastha, DISCO (Distributed

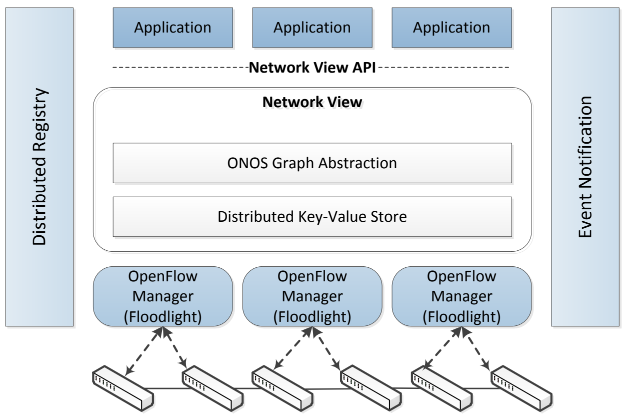


Fig. 7 ONOS Architecture (Berde et al., 2014)

Multi-domain SDN Controllers), ONOS, and Kandoo. Those controllers are designed to support fully distributed architecture and have the potential to support multi-domain SDN implementation.

ElastiCon (Dixit, Hao, Mukherjee, Lakshman, & Kompella, 2013) proposes a controller pool which dynamically grows or shrinks corresponding to traffic conditions, and the workload is dynamically allocated among the controllers.

Pratyaastha (Krishnamurthy et al., 2014) is an Efficient Elastic Distributed SDN Control Plane; it proposes a novel method for assigning SDN switches and partitions of SDN application state to distributed controller instances.

DISCO (Phemius et al., 2014) is another open and extensible SDN control plane architecture intended to address the distributed and heterogeneous characteristic of wide area networks and modern overlay networks. DISCO is implemented on top of Floodlight, an open source OpenFlow-enabled controller and consists of two parts: an intra-domain part where each controller manages its network domain, and an inter-domain part, which manages the communication with other DISCO controllers, through a lightweight control channel, to ensure end-to-end network services.

The Open Network Operating System (ONOS) (Berde et al., 2014) followed in the footsteps of previous SDN controllers such as Onix but is intended for release as an open source project which relies on the SDN community to contribute, examine, evaluate, and extend it. ONOS adopts the distributed architecture for high availability and scale-out. The ONOS characteristic abstract device approach means that the core operating system does not have to be aware of the particular protocol being used to control a device.

As shown in Fig. 7, in the distributed ONOS core, the ONOS instances work together to create what appears to the rest of the network and applications as a single platform. ONOS allows applications to examine the global network view and create flow paths that specify full or partial routes along with traffic that should flow over that route and other actions that should be taken, or use a global match-action (or match-instruction) abstraction which provides the full power of OpenFlow to enable an application to program any switch from a single vantage point.

OpenDaylight (ODL) (Jahan et al., 2014; Medved et al., 2014) makes it possible for the network to be logically and/or physically split into different slices or tenants. The split can be with parts of the controller, modules, explicitly dedicated

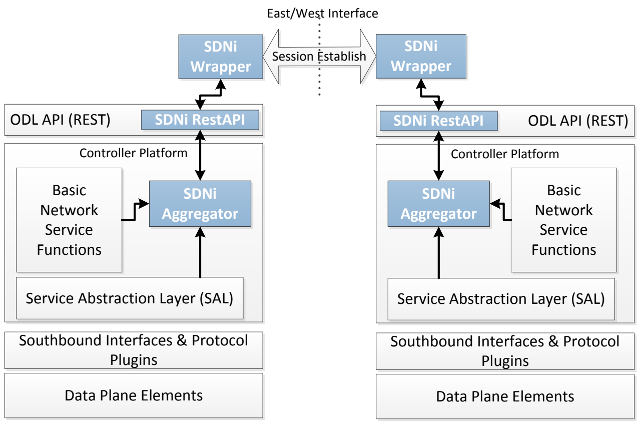


Fig. 9 OpenDaylight SDNi (Jahan, Shaik, Kotaru, & Kuppili, 2014)

to one or a subset of these slices. A multi-controller instance is developed by using an inter-SDN controller communication application called ODL-SDNi (Software Defined Network interface), which includes the SDNi Aggregator, a Northbound SDNi plugin that acts as an aggregator for collecting network information to be shared across federated SDN controllers, SDNi RestAPI, that is used to fetch the network information from the Northbound plugin, and SDNi Wrapper, which is responsible for collecting and sharing information to/from federated controllers. The ODL-SDNi architecture is shown in Fig. 9.

Kandoo (Hassas Yeganeh & Ganjali, 2012) is another distributed control plane design, which has a hierarchical architecture. Kandoo employs two layers of controllers, i.e. root controller (top layer) and local controller (bottom layer). Kandoo enables network operators to deploy local controllers on demand and relieve the load on the top layer controller, which is the only potential bottleneck regarding scalability.

Apart from the limited control plane architecture support for multi-domain SDN, the SDN interface, especially the Eastbound/Westbound interface, plays a significant role. SDN interconnect (SDNi), was among the first to connect SDN domains using an automated system (Yin et al., 2012). However, the SDNi draft form that was placed on the IETF data tracker in mid-2012 expired in December 2012; no further work was done, and it appears to have been abandoned. Another approach to the interconnection of SDN controllers is the West-East Bridge (Lin et al., 2015), which is an ongoing project. It has already successfully implemented an international test bed that connects heterogeneous controllers across four SDN domains (CERNET, CSTNET, Internet2, and SURFnet). Lastly, (Helebrandt & Kotuliak, 2014) proposed a novel multi-domain SDN architecture and defined an interconnection protocol. However, little information is currently available in the literature.

Hence, there are two implementation approaches for completely distributed controllers: the horizontal approach and the vertical approach, as shown in Fig. 8. In horizontally distributed controllers, multiple controllers are organized in a flat control plane where each one governs a subset of the network switches. This can be done either with state replication or without state replication. Vertically distributed controllers are a hierarchical control plane where the controller functionality is organized vertically. In this deployment model, control tasks are distributed to different controllers depending on selected criteria such as network

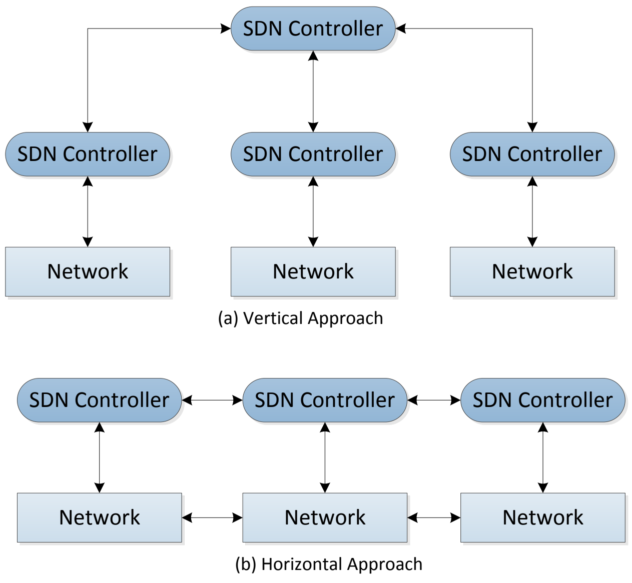


Fig. 8 Approaches in Distributed SDN Controller

view and locality requirements. Thus, local events are handled by the controller that is lower in the hierarchy, and global events are handled at the higher level.

4. Multi-Domain SDN Challenges

Despite the ongoing work being done on distributed SDN architectures and the SDN controller interconnection, several challenges remain. We will discuss some of those challenges and future works of multi-domain SDN below.

4.1. Controller Scalability

As mentioned in the previous section, the logically centralized control plane typically consists of multiple distributed controllers. The distributed controllers help to improve the number of flow requests handled per controller and to reduce the flow request response time. The logically centralized control plane architecture aims to maintain a consistent global network view and the controllers share information through a state synchronization mechanism which can overload the network due to the frequent changes that occur in the network. To improve scalability, the state synchronization load should be reduced to maintain a consistent information state among controllers.

A Load Variance-based Synchronization (LVS) method is proposed in (Guo et al., 2014) to cope with this challenge. LVS-based schemes conduct effective state synchronization among controllers only when the load of a specific network or domain exceeds a certain threshold. Therefore, the LVS can effectively reduce the synchronization overhead among controllers.

Another approach to achieve a global network view was proposed that utilizes local algorithms in a distributed control plane (Schmid & Suomela, 2013). The improvement is based on the reasoning that each controller needs to communicate only with its local neighbors, within a predefined number of hops. Accordingly, the controller load can be reduced and load balance among neighbouring controllers is achieved.

4.2. Controller Placement

The distributed control plane leads to open challenges, such as determining the number of controllers required. The number of controllers and their location has a direct impact on SDN performance. Characterization of the placement problem of controllers was formally introduced by Heller et al. (Heller et al., 2012). The authors consider the optimal controller placement problem in an attempt to minimize the propagation delay between controllers or controller-switch in the WAN. This work identifies open issues for future work.

Other work undertaken by (Jimenez et al., 2014) defines the principles for designing a scalable control plane so as to address the controller placement problem. They use an algorithm called k-Critica to find the minimum number of controllers and their locations to build a robust control network topology.

However, the methods discussed focus on static networks and do not support dynamic networks. In (Bari et al., 2013), a framework for dynamically deploying multiple controllers in the WAN is proposed. According to the network status, the number of required active and inactive controllers and their locations could be identified, and this work could effectively decrease the set-up time for traffic flows.

4.3. Multi-Domain SDN Communication Protocol

Communication between multiple controllers utilizing the eastbound/westbound interface is the important key component of multi-domain SDN architecture. Despite alternate approaches being proposed, the logical approach for IP-based computer networking is to use existing inter-domain protocols, as there remains a requirement to communicate with legacy network devices.

The peering between autonomous systems or domains in current global networks supporting the Internet typically utilize the Border Gateway Protocol (BGP). An comparative experiment highlighting the use of BGP in multi-domain SDN and an ordinary IP network is presented in (Wibowo & Gregory, 2016). This comparative study suggested BGP works in both type of network and despite its mediocre operational performance compared to existing IP multi-domain networks, SDN propounds the flexibility and more advance network management interaction. Therefore, enhancements in the eastbound/westbound interface to accommodate improved multi-domain solutions would be beneficial and provide an overall performance improvement for multi-domain SDN communication.

4.4. Discussion and Future Work

SDN offers improved control of a network by introducing a programmability capability and the separation of the control plane from the data plane. The potential benefits of these features are enhanced configuration, improved performance, and innovation in future network architecture design. With its ability to gather the instantaneous network status, SDN allows for the real-time centralized control of a network based on both the instantaneous network status and user-defined policies. This leads to benefits in optimizing network configurations and improving network performance. The potential benefit of SDN is further evidenced by the fact that SDN offers a convenient platform for the development of new techniques and encourages innovative network design, attributed to its programmability and the ability to define isolated virtual networks via the control plane. In addition to the benefits described, SDN facilitates virtualization, device configuration, and troubleshooting. The ONF describe several use cases for SDN such as campus networks, data centers, and Cloud computing.

This paper has provided a review of the SDN architecture for multi-domain networks and examines the significant work being done in the area of distributed SDN controller architecture supporting these networks. For the carriage service provider, a multi-domain SDN makes it possible to have optimum Capital Expenditure (Capex), and to reduce Operational Expenditure (Opex) while operating a network that might comprise both national and international networks. However, several issues still need to be tackled to improve the development and use of multi-domain SDN, especially in carriage service provider networks.

The horizontal and vertical approaches in a distributed SDN control plane can support multi-domain SDN. From the perspective of carriage service providers, factors such as network cost, QoS, and network performance still need to be analyzed. Scalability problems should also be studied, and this is an area of current research. The location and the number of controllers also significantly affects network performance. The coordination between controllers is an essential mechanism that warrants investigation. The Eastbound/Westbound interface still requires more work to find an efficient means of communication between SDN domains. An increase in the number of efficient interfaces will significantly enhance SDN development. Controller security is an important issue since attacks on the controllers could compromise the entire network, which spells disaster for a carriage service provider. The issues of controller security and the techniques to secure communication between controllers and controller switches are major issues that should be tackled in future work.

5. Conclusions

This paper provides a survey of the research literature pertaining to various approaches for a multi-domain SDN architecture. A description of the SDN architecture and its major elements, including OpenFlow, is provided as well as a comparative guide to selected controllers and their features. We classified the centralized and distributed SDN models, which leads to our understanding of the current status of research on multi-domain SDN. The key challenges that affect performance including scalability, number and placement of the controllers has been discussed.

As SDN is beginning to attract the attention of carriage service providers, the need for a multi-domain SDN solution, with scalability support, is one of the critical challenges. From the perspective of carriage service providers, research related to multi-domain SDN, including a comparative analysis of approaches to network cost, QoS and performance, will be very beneficial. Challenges remain including security, interoperability, and reliability.

References

Ahmed, R., & Boutaba, R. (2014). Design considerations for managing wide area software defined networks. *Communications Magazine, IEEE, 52*(7), 116-123. doi: 10.1109/MCOM.2014.6852092

Alexander, D. S., Arbaugh, W. A., Hicks, M. W., Kakkar, P., Keromytis, A. D., Moore, J. T., . . . Smith, J. M. (1998). The SwitchWare active network architecture. *Network, IEEE, 12*(3), 29-36. doi: 10.1109/65.690959

Award-winning Software-defined Networking NEC ProgrammableFlow® Networking Suite. (2013). Retrieved from: http://www.necam.com/docs/?id=67c33426-0a2b-4b87-9a7a-d3cecc14d26a

Bari, M. F., Roy, A. R., Chowdhury, S. R., Zhang, Q., Zhani, M. F., Ahmed, R., & Boutaba, R. (2013). *Dynamic controller provisioning in software defined networks.* Paper presented at the Network and Service Management (CNSM), 2013 9th International Conference on.

Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., . . . Snow, W. (2014). *ONOS: towards an open, distributed SDN OS.* Paper presented at the Proceedings of the third workshop on Hot topics in software defined networking.

Big Cloud Fabric: Hyperscale Networking for All. (2014). Retrieved from: http://go.bigswitch.com/rs/974-WXR-561/images/Big%20Cloud%20Fabric%20Data%20Sheet%203.0_WEB.pdf?_ga=1.231291420.225344052.1440643037

Braun, W., & Menth, M. (2014). Software-Defined Networking using OpenFlow: Protocols, applications and architectural design choices. *Future Internet, 6*(2), 302-336.

Brocade Vyatta Controller. (2015). Retrieved from: http://www.brocade.com/content/brocade/en/backend-content/pdf-page.html?http://www.brocade.com/content/dam/common/documents/content-types/datasheet/brocade-vyatta-controller-ds.pdf

Cai, Z. (2011). *Maestro: Achieving scalability and coordination in centralizaed network control plane.* ProQuest, UMI Dissertations Publishing.

Campbell, A. T., Katzela, I., Miki, K., & Vicente, J. (1999). Open signaling for ATM, internet and mobile networks (OPENSIG'98). *SIGCOMM Comput. Commun. Rev., 29*(1), 97-108. doi: 10.1145/505754.505762

Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., & Shenker, S. (2007). *Ethane: taking control of the enterprise.* Paper presented at the ACM SIGCOMM Computer Communication Review.

Casado, M., Garfinkel, T., Akella, A., Freedman, M. J., Boneh, D., McKeown, N., & Shenker, S. (2006). *SANE: a protection architecture for enterprise networks*. Paper presented at the Proceedings of the 15th conference on USENIX Security Symposium - Volume 15, Vancouver, B.C., Canada.

Cisco, C. V. N. I. (2015). Cisco Visual Networking Index: Forecast and Methodology, 2014-2019. *Cisco Public Information*.

Cisco Open SDN Controller 1.1. (2015). Retrieved from: http://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/open-sdn-controller/datasheet-c78-733458.pdf

Contrail Architecture. (2013). Retrieved from: http://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000535-en.pdf

Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., & Banerjee, S. (2011). DevoFlow: scaling flow management for high-performance networks. *ACM SIGCOMM Computer Communication Review, 41*(4), 254-265.

Dabbagh, M., Hamdaoui, B., Guizani, M., & Rayes, A. (2015). Software-defined networking security: pros and cons. *IEEE Communications Magazine, 53*(6), 73-79. doi: 10.1109/MCOM.2015.7120048

Dixit, A., Hao, F., Mukherjee, S., Lakshman, T., & Kompella, R. (2013). *Towards an elastic distributed SDN controller.* Paper presented at the ACM SIGCOMM Computer Communication Review.

Egilmez, H. E. (2014). Distributed QoS architectures for multimedia streaming over software defined networks. *Multimedia, IEEE Transactions on, 16*(6), 1597-1609.

Erickson, D. (2013). *The beacon openflow controller*. Paper presented at the Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, Hong Kong, China. http://delivery.acm.org/10.1145/2500000/2491189/p13-erickson.pdf?ip=131.170.90.6&id=2491189&acc=PUBLIC&key=65D80644F295BC0D%2E124032AC6F25F239%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=706563586&CFTOKEN=26197242&__acm__=1440464188_b9326b56763b433afeb5067a552ee140

http://delivery.acm.org/10.1145/2500000/2491189/p13-erickson.pdf?ip=131.170.90.6&id=2491189&acc=PUBLIC&key=65D80644F295BC0D%2E124032AC6F25F239%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=706563586&CFTOKEN=26197242&__acm__=1440464192_8016844fb178ce34c779b8a35c8eb031

Farhady, H., Lee, H., & Nakao, A. (2015). Software-Defined Networking: A survey. *Computer Networks, 81*, 79.

Floodlight, S. OpenFlow Controller. *Web:* https://github.*com/floodlight/floodlight*.

Foundation, O. N. (2012). Software-defined networking: The new norm for networks. *ONF White Paper*.

Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., . . . Zhang, H. (2005). A clean slate 4D approach to network control and management. *ACM SIGCOMM Computer Communication Review, 35*(5), 41-54.

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., & Shenker, S. (2008). NOX: Towards an operating system for networks. *ACM SIGCOMM Comp. Commun. Rev., 38*(3), 105-110.

Guo, Z., Su, M., Xu, Y., Duan, Z., Wang, L., Hui, S., & Chao, H. J. (2014). Improving the performance of load balancing in software-defined networks through load variance-based synchronization. *Computer Networks, 68*, 95-109.

Hakiri, A., Gokhale, A., Berthou, P., Schmidt, D. C., & Gayraud, T. (2014). Software-Defined Networking: Challenges and research opportunities for Future Internet. *Computer Networks, 75, Part A*, 453-471. doi: http://dx.doi.org/10.1016/j.comnet.2014.10.015

Haleplidis, E., Hadi Salim, J., Halpern, J. M., Hares, S., Pentikousis, K., Ogawa, K., . . . Koufopavlou, O. (2015). Network Programmability With ForCES. *Communications Surveys & Tutorials, IEEE, 17*(3), 1423-1440. doi: 10.1109/COMST.2015.2439033

Hassas Yeganeh, S., & Ganjali, Y. (2012). *Kandoo: a framework for efficient and scalable offloading of control applications.* Paper presented at the Proceedings of the first workshop on Hot topics in software defined networks.

Helebrandt, P., & Kotuliak, I. (2014, 4-5 Dec. 2014). *Novel SDN multi-domain architecture.* Paper presented at the Emerging eLearning Technologies and Applications (ICETA), 2014 IEEE 12th International Conference on.

Heller, B., Sherwood, R., & McKeown, N. (2012). *The controller placement problem.* Paper presented at the Proceedings of the first workshop on Hot topics in software defined networks.

HP VAN SDN Controller Software. Retrieved from: http://h20195.www2.hp.com/v2/getpdf.aspx/4AA4-9827ENW.pdf

Huawei IP SDN Controller Open and Application. (2013). Retrieved from: http://www.euchina-fire.eu/wp-content/uploads/2015/01/Huawei-IP-SDN-Controller-Open-and-ApllicationV1-2.pdf

IBM Programmable Network Controller V3.0. (2012).

Ishii, S., Kawai, E., Takata, T., Kanaumi, Y., Saito, S., Kobayashi, K., & Shimojo, S. (2012, 12-14 Dec. 2012). *Extending the*

RISE controller for the interconnection of RISE and OS3E/NDDI. Paper presented at the Networks (ICON), 2012 18th IEEE International Conference on.

Jahan, R., Shaik, S., Kotaru, K., & Kuppili, D. C. (2014). ODL-SDNi Retrieved December, 2016, from https://wiki.opendaylight.org/view/ODL-SDNi_App:Main

Jammal, M., Singh, T., Shami, A., Asal, R., & Li, Y. (2014). Software defined networking: State of the art and research challenges. *Computer Networks, 72*, 74-98. doi: http://dx.doi.org/10.1016/j.comnet.2014.07.004

Jarraya, Y., Madi, T., & Debbabi, M. (2014). A Survey and a Layered Taxonomy of Software-Defined Networking. *Communications Surveys & Tutorials, IEEE, 16*(4), 1955-1980. doi: 10.1109/COMST.2014.2320094

Jimenez, Y., Cervello-Pastor, C., & Garcia, A. J. (2014, 2-4 June 2014). *On the controller placement for designing a distributed SDN control layer*. Paper presented at the Networking Conference, 2014 IFIP.

Jinke, Y., Ying, W., Keke, P., Shujuan, Z., & Jiacong, L. (2016, 5-7 Oct. 2016). *A load balancing mechanism for multiple SDN controllers based on load informing strategy*. Paper presented at the 2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS).

Karakus, M., & Durresi, A. (2016, 23-25 March 2016). *A Scalability Metric for Control Planes in Software Defined Networks (SDNs)*. Paper presented at the 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA).

Kaur, R., Singh, A., Singh, S., & Sharma, S. (2016, 3-5 March 2016). *Security of software defined networks: Taxonomic modeling, key components and open research area*. Paper presented at the 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT).

Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., . . . Hama, T. (2010). *Onix: A Distributed Control Platform for Large-scale Production Networks*. Paper presented at the OSDI.

Kotronis, V., Gämperli, A., & Dimitropoulos, X. (2015). Routing centralization across domains via SDN: A model and emulation framework for BGP evolution. *Computer Networks, 92, Part 2*, 227-239. doi: http://dx.doi.org/10.1016/j.comnet.2015.07.015

Kreutz, D., Ramos, F. M. V., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., & Uhlig, S. (2015). Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE, 103*(1), 14-76. doi: 10.1109/JPROC.2014.2371999

Krishnamurthy, A., Chandrabose, S. P., & Gember-Jacobson, A. (2014). *Pratyaastha: an efficient elastic distributed SDN control plane*. Paper presented at the Proceedings of the third workshop on Hot topics in software defined networking, Chicago, Illinois, USA. http://dl.acm.org/citation.cfm?doid=2620728.2620748

Kubo, R., Fujita, T., Agawa, Y., & Suzuki, H. (2014). Ryu SDN Framework - Open-source SDN Platform Software. *12*. Retrieved from: https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201408fa4.pdf&mode=show_pdf

Lin, P., Bi, J., Wolff, S., Wang, Y., Xu, A., Chen, Z., . . . Lin, Y. (2015). A west-east bridge based SDN inter-domain testbed. *IEEE Communications Magazine, 53*(2), 190-197. doi: 10.1109/MCOM.2015.7045408

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., . . . Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review, 38*(2), 69-74.

Medved, J., Varga, R., Tkacik, A., & Gray, K. (2014). *Opendaylight: Towards a model-driven sdn controller architecture*. Paper presented at the Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014.

Nguyen, T. A., Eom, T., An, S., Park, J. S., Hong, J. B., & Kim, D. S. (2015, 18-20 Nov. 2015). *Availability Modeling and Analysis for Software Defined Networks*. Paper presented at the 2015 IEEE 21st Pacific Rim International Symposium on Dependable Computing (PRDC).

Nunes, B. A. A., Nguyen, X.-N., Turletti, T., Mendonca, M., & Obraczka, K. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys and Tutorials, 16*(3), 1617-1634. doi: 10.1109/SURV.2014.012214.00180

OneController™ OpenDaylight-based SDN Controller. (2015). Retrieved from: http://learn.extremenetworks.com/rs/extreme/images/OneController-DS.pdf

Phemius, K., Bouet, M., & Leguay, J. (2014). *Disco: Distributed multi-domain sdn controllers*. Paper presented at the Network Operations and Management Symposium (NOMS), 2014 IEEE.

The Real-Time Cloud. (2014). Retrieved from: http://www.ericsson.com/res/docs/whitepapers/wp-sdn-and-cloud.pdf

Saikia, D., & Malik, N. (2015). An Introduction to Open MUL SDN Suite.

Sallahi, A., & St-Hilaire, M. (2015). Optimal Model for the Controller Placement Problem in Software Defined Networks. *IEEE Communications Letters, 19*(1), 30-33. doi: 10.1109/LCOMM.2014.2371014

Schmid, S., & Suomela, J. (2013). *Exploiting locality in distributed sdn control*. Paper presented at the Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.

Shuhao, L., & Baochun, L. (2015). On scaling software-Defined Networking in wide-area networks. *Tsinghua Science and Technology, 20*(3), 221-232. doi: 10.1109/TST.2015.7128934

Singla, A., & Rijsman, B. (2013). OpenContrail Architecture Document. 2015, from http://www.opencontrail.org/opencontrail-architecture-documentation/

Tootoonchian, A., & Ganjali, Y. (2010). *HyperFlow: A distributed control plane for OpenFlow*. Paper presented at the Proceedings of the 2010 internet network management conference on Research on enterprise networking.

Vizarreta, P., Machuca, C. M., & Kellerer, W. (2016, 13-15 Sept. 2016). *Controller placement strategies for a resilient SDN control plane*. Paper presented at the 2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM).

Voellmy, A., & Wang, J. (2012). *Scalable software defined network controllers*. Paper presented at the Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication.

Wibowo, F. X. A., & Gregory, M. A. (2016). *Software Defined Networking Properties in Multi-Domain Networks*. Paper presented at the International Telecommunication Networks and Applications Conference 2016, Dunedin, New Zealand.

Xia, W., Wen, Y., Foh, C., Niyato, D., & Xie, H. (2015). A Survey on Software-Defined Networking. *IEEE Communications Surveys & Tutorials, 17*(1), 27-51. doi: 10.1109/COMST.2014.2330903

Yin, H., Xie, H., Tsou, T., Lopez, D., Aranda, P., & Sidi, R. (2012). SDNi: A Message Exchange Protocol for Software

Defined Networks (SDNS) across Multiple Domains. 2015, from https://datatracker.ietf.org/doc/draft-yin-sdn-sdni/

Yu, J., & Al Ajarmeh, I. (2010, 7-13 March 2010). *An Empirical Study of the NETCONF Protocol.* Paper presented at the Networking and Services (ICNS), 2010 Sixth International Conference on.

Yu, M., Rexford, J., Freedman, M. J., & Wang, J. (2010). Scalable flow-based networking with DIFANE. *ACM SIGCOMM Computer Communication Review, 40*(4), 351-362.