I.J. Wireless and Microwave Technologies, **2022, 2, 11-32**
Published Online April 2022 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijwmt.2022.02.02

SDN Interfaces: Protocols, Taxonomy and Challenges

Suhail Ahmad[1], Ajaz Hussain Mir[2]
E & C Engineering Department, National Institute of Technology Srinagar, J & K, 190006
Email: [1]suhail.sam008@gmail.com, [2]ahmir@rediffmail.com

Received: 09 November 2021; Revised: 29 December 2021; Accepted: 15 January 2022; Published: 08 April 2022

Abstract: The ever-increasing demands of Internet services like video on demand, big data applications, IoE and multi-tenant data centers have compelled the network industry to change its conventional non-evolving network architecture. Software Defined Network (SDN) has emerged as a promising network architecture which provides necessary abstractions and novel APIs to facilitate network innovations and simplifies network resource management by breaking the conventional network into multiple planes. All these SDN planes interact through open interfaces or APIs which are commonly categorized into southbound, northbound and west/eastbound interfaces. In this manuscript, we have identified and emphasized various communication protocols used at south and northbound interfaces. We have provided a taxonomy of south and northbound communication protocols based on their dependence, capabilities and properties. The pros and cons associated with each communication mechanism are highlighted and the numerous research challenges and open issues involved at these two interfaces are elucidated. In addition to it, we have proposed the necessary abstractions and extensions required in communication protocols at these two interfaces to simplify real-time monitoring and virtualization in next generation networks.

Index Terms: SDN, SDN Architecture, SDN Interfaces, OpenFlow, Northbound Interface, Southbound Interface

1. Introduction

In the last two decades, the use of Internet has increased drastically, with more than 4.5 billion users connected globally and in future 66% of world population is expected to be connected to Internet by the end of 2023 [1, 2]. This surge is mainly due to unprecedented growth of online services like big data applications, IoE, video on demand and cloud services. Consequent to it, the rigid conventional networks have been stretched to breaking points.

The inflexibility and rigidity in conventional networks are mainly due to closed, vendor-specific and vertically integrated network devices. The tight coupling of control and data plane in legacy devices hampers network innovations, dynamic programming and automatic reconfiguration of network infrastructure. Further, lack of centralized network view in conventional networks makes network management very challenging and error prone process which often leads to network bugs, faults and security breaches [3-5]. Due to these rigidities in conventional networks, the transition from IPv4 to IPv6 is still largely incomplete and a clean-slate approach to change Internet architecture is considered as a daunting task [6, 7].

In modern communication networks, network operators require agile, flexible and cost-effective network architecture to achieve dynamic and efficient resource utilization. Software Defined Network (SDN) has emerged as a promising networking paradigm for managing intricate and diverse network infrastructures [8, 9]. It breaks vertical integration by separating control and data plane, thus removes obstacles for innovations in both planes. This separation makes forwarding devices simple and network control logic can be implemented in a specialized centralized SDN controller [10]. However, to mitigate scalability and reliability issues the control logic can be implemented on several physically distributed but logically centralized servers [11]. The centralized network control in SDN provides bird-eye view of the entire network, simplifying job of the network manager by providing a single central interface for policy enforcement and reconfiguration. The conventional and SDN networking paradigms are shown in figure 1, and a comparison between them is drawn in table 1.

In brief, SDN is an agile, programmable and centrally managed network architecture which supports vendor-agnostic open devices. To meet varying traffic demands in next generation networks, the logically centralized SDN controller enables network administrators to dynamically regulate and load balance network-wide traffic flows. This networking paradigm enables creation of dynamic network topologies in data centers and policy-based routing in enterprise or service provider's network. The SDN interfaces play a vital role in realizing these goals and in this manuscript our aim is to identify various communication protocols used at south and northbound interfaces with emphasis on open issues and prospective trends which may drive future research in SDN communication protocols.

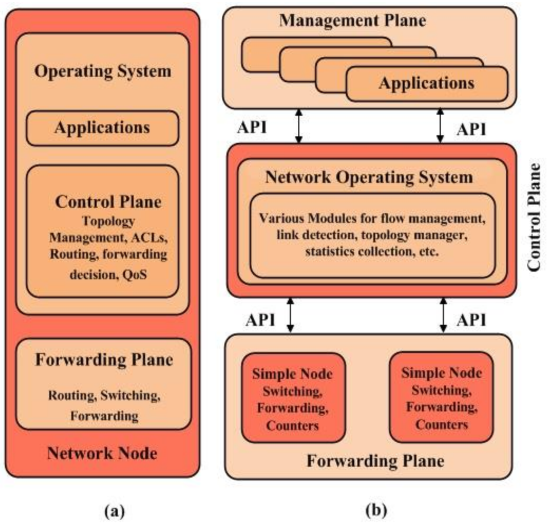
Operating System
Applications
Control Plane
Topology Management, ACLs, Routing, forwarding decision, QoS
Forwarding Plane
Routing, Switching, Forwarding
Network Node
Management Plane
Applications
API
API
Network Operating System
Various Modules for flow management, link detection, topology manager, statistics collection, etc.
Control Plane
API
API
Simple Node Switching, Forwarding, Counters
Simple Node Switching, Forwarding, Counters
Forwarding Plane
(a)
(b)

Fig. 1. Conventional network compared with SDN: (a) Conventional Network Node; and (b) Software Defined Network

Table 1. Comparison of Conventional Networks with Software Defined Networks

Features	Conventional Networks	Software Defined Networks
Communication Nodes	Vendor Specific, COTS with inbuilt Control Plane	Simple forwarding hardware either physical or virtual
Control Plane	Distributed control Distributed computation for routing Proprietary and closed OS Limited support for configuration or extension	Logically centralized control Centralized view for efficient resource management Open and customizable OS Highly extendible and configurable
Performance	Heterogeneity in network devices hampers the overall optimization of resources	Monitoring and management at a single central point enables better performance
Configuration	Tedious and error prone manual configuration	Based on overall network status, the centralized controller configures the data plane
Scope of Innovation	Closed and vendor specific hardware prevents modification for experimentation and innovations	With programmable and open devices, SDN encourages new ideas, services and innovations
Support for Virtualization	Limited support	Complements network virtualization and NFV

1.1 Contributions

The primary focus of this manuscript is on SDN interfaces particularly south and northbound SDN interfaces. Numerous protocols have been proposed for these two interfaces and each protocol provides diverse set of functions and capabilities. In this manuscript, we have attempted to analyze all these communication mechanisms used at these two interfaces with emphasis on identification of immediate requirements and abstractions to support real-time monitoring and virtualization in next generation networks. The contributions made are as follows:

- *Overview of SDN Interfaces:* The background information about multilayered SDN architecture is presented and an overview of SDN interfaces is provided.
- *Southbound Interface:* The most popular southbound standard OpenFlow [12] is discussed along with the improvements made to it in different versions. We have analyzed various other solutions or protocols used at this interface and have classified them into two broad categories: OpenFlow dependent and OpenFlow independent. In addition to it, we have highlighted the issues associated with each protocol and have compared these protocols based on their merits and features.
- *Northbound Interface:* We have classified the communication mechanisms used at northbound interface into following three categories: controller-based, intent based, and goal oriented. All such protocols/communication mechanisms are analyzed in detail and a comparative analysis is also provided.
- *Protocols used in diverse SDN controllers:* The communication mechanisms used at south and northbound interface in diverse SDN controllers are identified and expounded.
- *Proposed abstractions/extensions and future research perspectives:* We have specified the potential future research directions in both south and northbound interfaces. The problems associated with existing solutions for these two interfaces are highlighted along with the recommendations on probable solutions. We have proposed the necessary extensions and abstractions required at these two interfaces to support real-time network

monitoring and virtualization in next generation networks. Further, the security issues associated with these two interfaces and privacy concerns in network virtualization are highlighted.

1.2 Related Work

The network *programmability* and *adaptability* resurfaced as top research trends in computer networking immediately after the introduction of OpenFlow [12]. Afterwards, the SDN paradigm was widely accepted and implemented in diverse domains to achieve centralized network control, programmability and adaptability in modern communication networks. There have been numerous surveys [13-21] addressing different facets of SDN paradigm including SDN's historical view [14,15], SDN architecture, design issues, implementation and applications [16-22], fault management in SDN [23, 24], traffic engineering with SDN [25, 26], SDN Controllers [27, 28], and security issues in SDN [29-31].

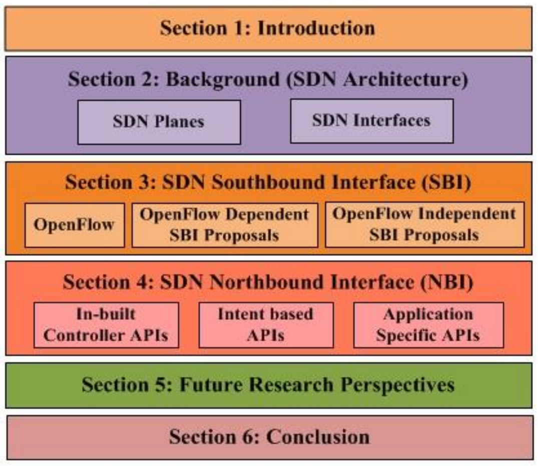
Section 1: Introduction
Section 2: Background (SDN Architecture)
SDN Planes
SDN Interfaces
Section 3: SDN Southbound Interface (SBI)
OpenFlow
OpenFlow Dependent SBI Proposals
OpenFlow Independent SBI Proposals
Section 4: SDN Northbound Interface (NBI)
In-built Controller APIs
Intent based APIs
Application Specific APIs
Section 5: Future Research Perspectives
Section 6: Conclusion

Fig. 2. Paper organization

The SDN interfaces which provide necessary abstractions and play an important role in achieving benefits from this radical networking architecture has been discussed as a section in few papers [13, 18]. Despite numerous surveys, none of the manuscript has discussed in detail the protocols proposed for SDN interfaces. This manuscript provides a systematic and extensive survey of south and northbound SDN communication protocols with emphasis on the pros and cons of the proposed approaches.

1.3 Paper Organization

This manuscript is organized into five sections, as depicted in figure 2. Section 2 provides an overview of SDN architecture and introduces various SDN interfaces. In section 3, the various versions of OpenFlow standard are presented and OpenFlow dependent and independent southbound protocol proposals are discussed. Section 4 provides the classification of northbound communication protocols along with their pros and cons. The future research perspectives and abstractions required for network monitoring and virtualization are highlighted in section 5 and finally the paper is concluded in section 6.

2. Background

In this section, we have presented the multi-layered SDN architecture and the relevant terminology. The SDN interfaces which enable segregation of management and control logic from forwarding functions and play a vital role in realizing benefits from this radical networking architecture are also expounded in this section.

2.1 SDN Planes

The Open Network Foundation (ONF) [32] is a leading standardization body for SDN which is supported by more than hundred organizations. It has proposed triple-layered SDN architecture to simplify network management, network programmability and automation of network operations. The multi-layered SDN architecture is depicted in figure 3, comprising of following three planes:

i. Data Plane: The forwarding plane or data plane consists of distributed forwarding devices usually software or hardware switches which forward packets as per the flow rules installed in their flow tables by the centralized SDN controller. These devices remain connected with the centralized SDN controller over a secure channel and are very often programmed with the help of OpenFlow protocol.

ii. Control Plane: The central layer or control plane acts as a bridging layer between management and data plane. It implements network control logic to manage and control the data plane devices. The network control logic can be implemented either in a physically centralized SDN controller or physically distributed but logically centralized SDN controllers [27]. The SDN controller maintains up-to-date network state information and provides a single central entity for network configuration. It provides bird-eye view of the entire network topology to the management applications and enables them to specify high-level network policies without bothering about the low-level implementation details.

iii. Management Plane: The top layer or management plane consists of diverse SDN applications which implement specific network control and management strategies like traffic engineering (TE), load balancing, firewalls, etc. With the help of these distinct applications, the benefits of this networking paradigm can be realized in diverse domains ranging from small home or campus networks to multitenant data centers. Unlike conventional networks, implementing complex management strategies in SDN is quite simple as the network operator can implement intricate traffic regulating applications in the management plane without bothering about complicated control logic implementation details.

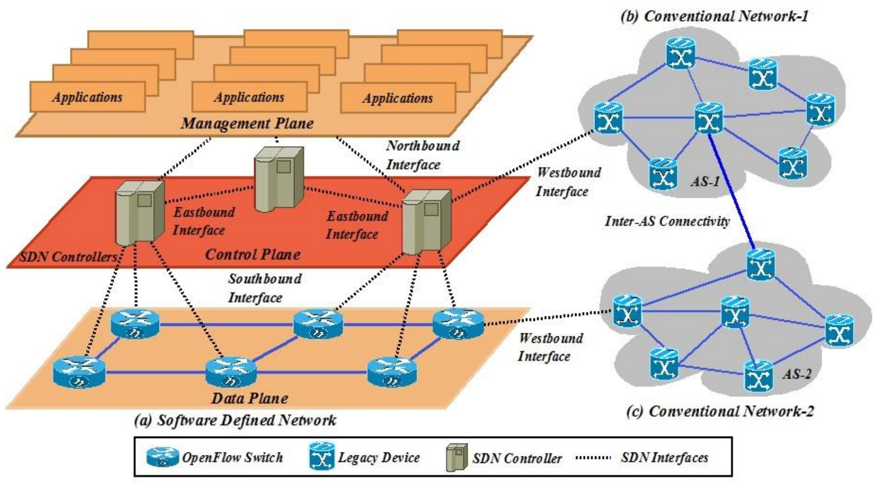

Fig. 3. Multi-layered SDN architecture showing: SDN Planes and SDN Interfaces

2.2 SDN Interfaces/APIs

The information exchange between the SDN planes and legacy distributed control takes place over the SDN interfaces/APIs, as shown in figure 3. In this sub-section, all these interfaces are elucidated along with their important functions.

i. Southbound Interface (SBI): The open vendor-agnostic interface between control and data plane is termed as southbound interface. It provides a generalization of device functionality to the controller and at the same time the controller uses it to communicate flow rules and retrieve flow statistics information from the data plane devices. In brief, it allows the SDN controller to discover network topology, communicate flow rules to switches, retrieve flow statistics from switches and implement policies defined by the applications in the management plane. The major challenges at this interface include heterogeneity in network devices and supported protocols. Numerous protocols are used at southbound interface but the most widely accepted industry standard is OpenFlow [12, 33].

ii. Northbound Interface (NBI): The applications in the management plane use NBI to interact with the control plane. The NBI can be compared with win32 or POSIX standard in operating systems as it provides necessary abstractions that guarantee programming language and controller platform independence. The SDN controllers support various programming languages and offer a wide range of northbound APIs. The researchers are focused towards a common northbound API but it is too early to have a single such standard. However, open and standard northbound API is very critical for application portability, interoperability and overall network management. Keeping in view the importance of this interface, the ONF formulated a working group namely Open Networking Foundation North Bound Interface Working Group (NBI-WG) [34] for standardization of this API.

iii. East/Westbound Interface: SDN provides centralized control over the forwarding devices, however, if a network scales-up, a single central controller can become a bottleneck [27, 35]. In large-scale networks, multiple SDN controllers are used to address scalability and reliability issues wherein each domain specific SDN controller handles a set of data plane devices. These SDN controllers share network information of their respective domains with each other to have a global network view. This information exchange involves east and westbound interfaces as shown in figure 3. As per the authors in [23], the westbound interface is used between two SDN controllers whereas the eastbound interface is used to exchange information between the SDN control plane and the legacy distributed control plane. Both east and westbound interfaces are yet to be standardized and various controller vendors use their own inbuilt proprietary APIs.

3. SDN Southbound Interface

The SBI allows separation of network control logic from forwarding functions in SDN. For efficient network operation and monitoring of forwarding devices, the SBI should be robust and secure. Other than OpenFlow, there are different SBI proposals which we have classified into two broad categories as OpenFlow dependent and OpenFlow Independent SBIs (as shown in figure 4). In this section, first we have provided a detailed analysis of OpenFlow and then the two categories of SBI protocol proposals are presented.

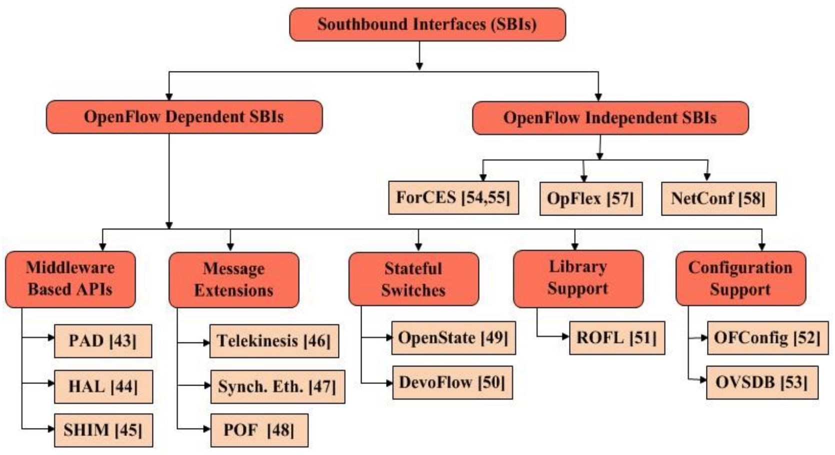

Fig. 4. Classification of SBI protocols

3.1 OpenFlow

OpenFlow is the most widely deployed SBI for SDNs. Since inception, OpenFlow has gone through various modifications. The basic version 1.0 [36] provided only a single flow table with 12 fixed header fields whereas the latest version 1.5 [41] added number of functions and includes 41 matching fields. The evolution of OpenFlow along with the most prominent features added with each version are shown in table 2.

OpenFlow version 1.0 has three components in a single flow table including header fields, actions and counters. Switches supporting this version can perform only one operation at a time due to single flow table, hence provides limited flexibility and scalability. On the other hand, OpenFlow version 1.1 introduced packet processing pipeline with the help of multiple flow tables. The Forwarding Abstraction Working Group (FAWG) [42] proposed Negotiable Datapath Model which provides abstraction of forwarding model expressed in terms of JSON. OpenFlow 1.1 renamed header field as match field and also introduced four types of group tables (All, Select, Indirect and Fast Failover) for specialized functions.

OpenFlow version 1.2 introduced Type-Length-Value (TLV) structure also referred as OpenFlow eXtensible Match (OXM), to provide more flexibility as compared to fixed match structure of previous version 1.1. Support for IPv6 was added in version 1.2 and controller failure or control plane availability issues were addressed by enabling a switch to connect to multiple controllers either in master/slave or equal mode. On the other hand, to provide QoS in SDN, OpenFlow version 1.3 introduced meter table. The meter table comprises of multiple meter entries, with each entry identified by a meter identifier.

For scalability and having support for unidirectional and bidirectional tables, OpenFlow version 1.4 introduced synchronized table. This version also included bundle which allows performing modifications on a group of forwarding switches. On the other hand, version 1.5 extended bundles to provide synchronization among numerous switches. Unlike previous versions which allow matching only on ingress packets, OpenFlow version 1.5 introduced Egress table which allows packet matching on output port. All these OpenFlow versions along with their main highlights are summarized in table 2.

Table 2. OpenFlow versions along with most prominent features

Version/ year/ Literature	Match Fields	Statistics	Flow Table /Group Table	Simultaneous communication with multiple controllers	Main Highlights
1.0, 2009 [36]	Ingress Port	Per table	Single/No	No	One table, only fixed header fields
	Ethernet: source, destination, type, VLAN	Per flow			
	IPv4: source, destination, protocol, ToS	Per port			
	TCP/UDP: source port, destination port	Per queue			
1.1, 2011 [37]	Metadata, SCTP, VLAN tagging	Group	Multiple/ Yes	No	Multiple and group tables MPLS and VLAN support
	MPLS: label, traffic class	Action bucket			
1.2, 2011 [38]	OXM using TLV		Multiple/ Yes	Yes	OXM, IPV6 Support, controller fault tolerance
	IPv6: Source, destination, flow label, ICMPv6,				
1.3, 2012 [39]	IPv6 Extension headers	Per-flow meter	Multiple/ Yes, more flexible	Yes, auxiliary connections enabled	Meter table, explicit table miss entry
1.4, 2013 [40]		Optical port properties	Multiple/ Yes, more flexible	Yes, auxiliary connections enabled	Table synchronization and bundle support
1.5, 2015 [41]	Output Port		Multiple/ Yes, more flexible	Yes, auxiliary connections enabled	Scheduled bundles and egress table

3.2 OpenFlow Dependent Southbound Interface Proposals

The OpenFlow dependent communication protocols are basically extensions in OpenFlow to extend centralized control over legacy devices or to address certain limitations of OpenFlow. We have classified these proposals into five categories as shown in figure 4. The middleware-based approaches introduce an intermediate layer between legacy device and the controller, to have centralized control over legacy distributed devices. In message extension approaches, the OpenFlow messages are extended or modified, to control conventional devices or to make data plane protocol oblivious. The third category includes those approaches which try to reduce controller-switch communication overhead and propose shifting of limited control back to data plane (stateful data plane). For library and configuration support, few approaches have been proposed. In this sub-section, all these proposals are discussed in detail along with their pros and cons.

A. Middleware based Proposals

Programmable Abstraction Datapath (PAD) [43] exposes switch capabilities to enable programmability in non-OpenFlow devices. More specifically, it is a hardware abstraction (programming model) for data plane devices that allows programming of data path behavior using generic byte operations by defining protocol headers and providing function definitions. The PAD architecture comprises of several components including ports, search engine, search structures, an execution engine and forwarding functions. In this architecture, when a packet arrives at incoming port with metadata, it is processed by the search engine which adds a function name to it. The appended function is then executed on the packet at the execution engine and based on the execution outcome the packet is forwarded to a particular output port for transmission. PAD provides a flexible solution for optical flows, which allows handling and changing frames, packets or datagrams while in the transit.

Hypertext Application Language (HAL) [44] is another middleware-based solution that attempts to control vendor-specific devices like OpenFlow compliant devices. HAL hides vendor specific features and device complexity by decoupling management logic and device-specific control from the node. This decoupling is achieved by defining two sub-layers: Hardware Specific Layer and Cross Hardware Platform Layer. The former is used to determine the particular hardware using hardware specific module whereas the latter provides node abstraction, interoperability by performing device required configurations. These two layers communicate to hide device complexity and heterogeneity. Similar to HAL is Hardware Shim [45], in which a traditional device is connected to an intermediate Shim device that enables it to communicate routing information to the controller and in return processes the OpenFlow messages. The OpenFlow messages communicated by the centralized SDN controller are translated into device specific configuration messages by the intermediate Shim hardware. All these approaches are limited in functionality and can handle a particular set of legacy devices. To handle diverse set of data plane devices, a more flexible and scalable approach is required, however such a solution may result in complex implementation.

B. OpenFlow Message Extension/Modification Proposals

Telekinesis [46] introduces a framework to control both SDN and legacy switches in a hybrid network. It introduces a new flow control primitive termed as LegacyFlowMod to modify flow table entries in legacy switches.

With the help of LegacyFlowMod control primitive, the controller influences mac-learning mechanism of layer 2 and instructs an OpenFlow switch to send a special packet to legacy switch, in order to modify legacy device forwarding table entries. This modification enables the legacy devices to forward the newly encountered data packets to nearest OpenFlow switch. Two algorithms are proposed by the authors in [46] which include path verification and path update. The former is used by the controller to check path feasibility and once feasible, the later is used to instruct OpenFlow switches to make necessary changes in legacy device forwarding table using LegacyFlowMod control primitive. The proposed approach is applicable only in layer 2 networks and may result in topological loops in the network. Further, it does not support other networking protocols and features like ACL violations, dynamic flow changes, etc.

Like Telekinesis, authors in [47] propose extensions to OpenFlow messages to control and manage synchronous Ethernet technology with the help of a centralized controller. An OpenDayLight controller bundle named SyncE Manager has been created in [47] to implement all the intelligence of Synchronous Ethernet operations at the controller. The SyncE works as an application over the controller and exchange information with the following controller applications: statistics manager, switch manager, topology manager and forwarding rules manager.

The researchers in [48] have pointed out certain deficiencies in OpenFlow like it is reactive rather than proactive and data plane needs to be protocol sentient in order to process incoming data packets. More precisely, data plane in OpenFlow is stateless (dependent on controller) and there is a parsing burden on it, as it has to understand and extract required bits of protocol headers in a specific format and match with the flow table entries communicated by the centralized controller. Further, with the advent of new versions of OpenFlow more header fields are introduced, leading to increased complexity and compatibility issues among different OpenFlow versions. The authors have proposed Protocol Oblivious Forwarding (POF) in which the packet parsing takes place at the controller and based on the parsing the controller generates sequence of generic keys and table lookup instructions that are installed in the data plane elements with the help of Flow Instruction Set (POF-FIS). Hence, parsing burden is on the controller and data plane devices appear as white boxes with only processing and forwarding capabilities and the entire behavior of the device lies under the control of the controller. Apart from POF, all other approaches discussed in this sub-section are having limited functionality and can work with a particular set of devices. On the other hand, one major limitation in POF is increased communication overhead between controller and switch.

C. *Stateful Switch Proposals*

The authors in [49] argue that making switches completely stateless and shifting entire network control logic into a centralized controller increases communication overhead between forwarding devices and the SDN controller. They have proposed OpenState, an architecture in which programmer installs local level functions into the forwarding devices using Extended Finite State Machine (EFSM). EFSM acts as an extension of OpenFlow's match-action mechanism which allows implementation of various local stateful tasks like MAC learning, port knocking, etc, within the forwarding device.

Likewise, authors of DevoFlow [50] state that the communication overhead between control and data plane in SDN is manifold. To address this issue, they have suggested devolving control of most flows back to forwarding devices and the controller manages only significant and targeted flows. This reduces the controller-switch communication and also makes efficient utilization of TCAM. Further, they have suggested collection of aggregated traffic flow statistics from the data plane so that the SDN controller can have better network visibility. However, both approaches [49, 50] require significant changes in switch design which involves high cost.

D. *OpenFlow Library Support Proposals*

Since inception, OpenFlow has gone through numerous changes as already discussed in section 3.1. Consequent to such modifications in OpenFlow versions, the third-party hardware and software in control and data plane requires massive changes. Further, to generate libraries for each and every version of OpenFlow is time consuming. Revised OpenFlow Library (ROFL) [51] addresses this problem by proposing an abstraction layer that hides the details of OpenFlow versions, providing a uniform API for developers and thus eases the application development process. A framework termed as eXtensible Datapath daemon (xDPd) is used in ROFL to define datapath elements. It uses three libraries: ROFL-common, ROFL-HAL and ROFL-pipeline. ROFL-common comprises of protocol parsers, message meld features which can be used to construct OpenFlow Endpoints (OFEs). Such OFEs hide OpenFlow protocol versions by mapping protocol wire representation to a set of programming classes which can be incorporated either at the controller end or at the data plane device. ROFL-pipeline can be used as a data model, state manager or as a software packet processing library which uses packet processing APIs to process packets in software or hardware-cum-software datapath elements. ROFL-HAL implements an interface termed as Abstract Forwarding API between hardware-dependent platform drivers and hardware-independent control and management modules.

E. *Configuration Support Proposals*

The controller uses OpenFlow to install flow rules into forwarding tables of data plane devices. However, it does not provide configuration and management services which are required to dispense IP addresses or any other necessary configuration information. In conventional networks, such tasks are being performed using either SNMP or manually by using CLI. In SDN, following two protocols provide configuration features: OpenFlow-Configuration protocol (OF-Config) [52] and OvSDB [53].

OF-Config [52] acts as a companion of OpenFlow protocol and allows remote configuration of forwarding devices. It enables controller to modify and configure data plane devices by using set of rules. Likewise, OvSDB [53] is a management protocol used to control traffic in virtual environments. OvSDB was created by Nicira but was later acquired by VMware. Originally, it was part of OpenvSwitch, which is an open source virtual switch, designed for Linux-based hypervisors supporting various features. It offers networking functions like creation of multiple virtual switch instances by control elements, attach interfaces to switches, set QoS parameters on interfaces, configure tunnels, manage queries and collect statistics. Thus, it acts as a complementary protocol to OpenFlow for OpenvSwitch.

3.3 OpenFlow Independent Southbound Interface Proposals

Prior to OpenFlow, there were certain proposals which supported logical centralization of network control with different scope. We have classified them as OpenFlow independent SBI APIs and in this sub-section, we have identified and analyzed three such proposals.

A. Forwarding and Control Element Separation (ForCES)

ForCES [54, 55] is a flexible approach for network management without changing the conventional network architecture. ForCES architecture is depicted in figure 5 (a). Like OpenFlow, ForCES also proposed data and control plane separation but in the same element rather than having an external entity as logically centralized controller. The two segregated elements in the same device are termed as forwarding and control element. Inside forwarding elements there are Logical Functional Blocks (LFBs) which perform a specific function. LFBs enables control element to control forwarding elements, perform logical computation and well-defined actions on the incoming packets. Unlike OpenFlow, which uses TCP as a transport protocol between control and data plane, ForCES supports SCTP for better reliability and security. An in-depth comparison of ForCES and OpenFlow is provided in [56]; however, the major differences in these two approaches can be summarized as follows:

i. To use OpenFlow, switches have to use a predefined specification, but ForCES can be deployment without such restrictions.
ii. For each new feature, OpenFlow needs modeling and standardization whereas ForCES support extensibility without re-standardization.
iii. Basic determinant for forwarding in OpenFlow is match and actions while as ForCES uses LFBs.
iv. Transport protocol used in OpenFlow is TCP whereas SCTP is used in ForCES.

B. OpFlex

OpFlex [57], in contrary to OpenFlow, tries to distribute complexity of managing network back to the data plane in order to improve scalability. OpFlex supports declarative control, a new mechanism to transfer abstract policy from a policy controller to a set of smart devices capable of rendering abstract policy. OpFlex architecture comprises of multiple repositories and OpFlex protocol as shown in figure 5 (b). The controller in OpFlex stores network policy in a policy repository and communicates centralized policy to policy element with the help of OpFlex protocol. OpFlex is an open and extensible protocol for transferring abstract policy in XML or JavaScript Object Notation (JSON) between a controller and a data plane device, including hypervisor switches, physical switches, etc. These end points after registration in end-point registry remain connected with policy element. One more repository used in OpFlex is Observer repository which stores all the network events and faults. The major limitation of this approach is lack of network programmability and dynamic network control.

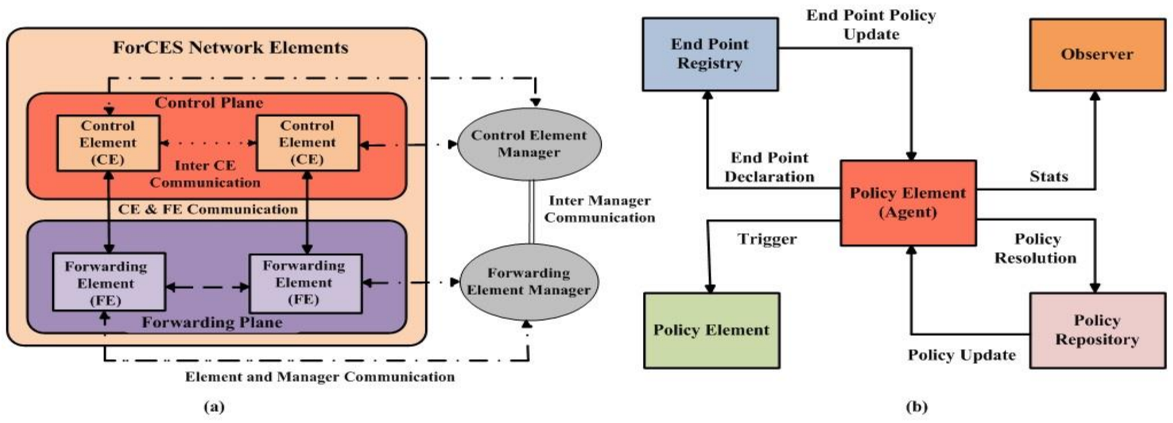

Fig. 5. (a) ForCES architecture, (b) OpFlex architecture

C. *NetConf*

Another proposal that was there prior to OpenFlow is NetConf [58]. It uses RPC based mechanism to install, modify and retrieve configuration information from network devices. The major advantage of NetConf is that it mirrors functionality of device supported management protocols, thus decreases cost and provides quick access to new features. It enables network applications to send or retrieve configuration datasets. It uses XML based data encoding for configuration data and protocol messages.

Summary: The SBI proposals discussed in this section are summarized in table 3. Out of these SBI proposals, OpenFlow and ForCES are two mature solutions, with the former being most widely accepted standard while as the later even though supported by IETF, failed to gain much attraction from the industry. ForCES facilitates extensibility, and is interoperable with conventional devices, but major limitation is lack of open source support. On the other hand, OpenFlow received overwhelming support from hardware vendors and programming community. However, OpenFlow also faces certain limitations like lack of upper-layer (L5-L7) header support and abstractions defined in it are difficult to implement in hardware. In OpFlex, there is no dynamic network control as it lacks network programmability. Likewise, NetConf lacks flexibility as it was designed with a different intent.

Table 3. Summary of proposals for SBI

OpenFlow Dependent SBI Proposals		
A. Middleware based APIs		
Southbound API	**Mechanism**	**Main Highlights**
PAD [43]	Generic Byte Operation	PAD allows programming of datapath behavior using generic byte operations, defining protocol headers and providing function definitions.
HAL [44]	Hardware Specific Layer and Cross Hardware Platform layer	HAL is a mediator that enables a southbound interface like OpenFlow to control heterogeneous devices.
Shim [45]	FPGA based Shim Hardware	Shim hardware which acts as intermediate device between legacy device and a SDN controller.
B. OpenFlow Message Extensions/Modifications		
Telekinesis [46]	LegacyFlowMod	It introduces a new flow control primitive termed as LegacyFlowMod to modify flow table entries in both SDN and legacy switch.
SyncE [47]	SyncE Manager	OpenFlow extensions to operate Synchronous Ethernet technology with a SDN Controller.
POF [48]	Flow Instruction Set (FIS)	Packet parsing takes place at the controller and based on the parsing, controller generates sequence of generic keys and table lookup instructions that are installed into the data plane.
C. Stateful Switches		
OpenState [49]	eXtensible Finite State Machines added to data plane devices	OpenState proposes extension of the OpenFlow match-action abstraction by extended finite machines that allow implementation of numerous tasks inside the data plane devices.
DevoFlow [50]	Shift limited control to switch	Minimizes the controller and data plane communication overhead at the cost of changes in switch design.
D. OpenFlow Library Support		
ROFL [51]	Using eXtensible DataPath daemon to support new OpenFlow versions	ROFL proposes an abstraction layer that hides the details of various OpenFlow versions to the application developer.
E. Configuration Support Protocols		
OFConfig [52]	Configuration of OpenFlow switches	It is an OpenFlow companion protocol which allows remote configuration of data plane devices
OVSDB [53]	OpenvSwitch management protocol	OVSDB is a management protocol developed as a part of OpenvSwitch, which is a feature rich open source virtual switch.
OpenFlow Independent SBI Proposals		
ForCES [54, 55]	Logical Function Block (LFB)	ForCES provide a flexible approach of separating control and data planes by enabling the separation in the same network element.
OpFlex [57]	Repositories and OpFlex protocol	OpFlex supports declarative control and is an open and extensible policy protocol for transferring abstract policy in XML or JavaScript Object Notation (JSON) between a controller and a data plane device.
NetConf [58]	Remote Procedure Call (RPC)	Mirrors device supported protocols, reduces cost and provides fast access to new features.

4. SDN Northbound Interface

The SDN NBI provides necessary abstractions to applications and hides complexity of underlying control logic from management plane. At SBI, the widely accepted API is OpenFlow but the same is not true for NBI, as there are number of approaches which are being worked out by the research community [59]. The researchers are focused towards a common northbound API but it is too early to have a single such standard, mainly due to different data models used in various control frameworks and variations in application requirements. However, open and standard northbound API is very critical for application portability and interoperability among different control platforms. The application development in SDN is arduous due to absence of a standard NBI. Various NBI APIs have been proposed and we have classified them into following three categories: i) In-built Controller APIs; ii) Intent based APIs and; iii) Goal-oriented APIs.

The in-built controller APIs either use ad-hoc, low-level, proprietary APIs tightly coupled with the controller platform or the REST architecture in which external applications (clients) use services provided by the controller (server). On the other hand, in Intent based approach, the application requirements are expressed in natural language and an SDN controller using its core functionality provides desired services. The requirement specific APIs or goal-oriented APIs try to simplify the bi-directional communication at NBI as per the intended requirements. In this section, we have presented all the three categories of NBIs along with their features and deficiencies.

4.1 In-built Controller APIs

Consequent to absence of standard NBI, various controller (centralized and distributed) platforms use in-built, adhoc and controller specific approaches for NBI. The centralized SDN controllers including NOX [60], POX [61], Trema [62], ParaFlow [63], Ryuo [64] and Rosemary [65] use in-built adhoc NBIs.

In NOX, the designers have mainly focused on functionality and have not considered the numerous practical considerations which are important for safety and isolation of network applications. The applications are developed in native core controller modules and a malicious or faulty application can overwrite the random memory or in the worst case can hang the entire system within an infinite loop. POX [61] is a python variant of NOX which inherits the features of NOX, hence, faces the same issues as that of NOX. On the other hand, Trema [62] provides a test-driven development framework but fails to provide abstractions that will enable easy and efficient development of SDN applications.

In ParaFlow [63], the main focus is on the concurrent execution of multiple applications using parallel event processing by multiple event handlers. ParaFlow is basically a lightweight controller written in C++ using Boost [66] library to achieve parallel asynchronous I/O. Like other centralized SDN controllers, it also employs adhoc API as NBI. Ryuo Controller [64] proposes a scalable architecture for SDN by reducing number of OpenFlow messages between control and data plane. The authors have introduced switch level services in the forwarding devices and such switch level services can use different southbound-APIs, however, these local services provide northbound-APIs for SDN applications. The Ryuo applications communicate with local services by using Pyro4 which is an open source implementation of RPC library for Python.

In contrast with aforesaid controllers, Rosemary [65] proposes a secure architecture which provides process containment, application permission structure and resource utilization monitoring. This secure architecture prevents stumbling of SDN controller operations in case of network application failure. It implements a resilient strategy and concept of network application containment based on the notion of spawning applications separated within a micro-NOS. At the NBI, it employs sandbox approach for access control and authentication of applications to prevent access of internal data structures and modify them without restriction by the malicious applications.

On the other hand, some controllers including Floodlight [67], Ryu [68], MUL [69], OpenDayLight [70] support RESTful APIs as NBI. The RESTful APIs [71] are based on REST software architecture, an alternative to SOAP and JavaScript Object Notation (JSON), which uses a set of principles that describe how distributed network resources are characterized by state and functionality. It is a layered architecture in which every resource is uniquely identifiable and communication between end entities takes place as client and server. Some distributed controllers also employ in-built NBIs like Onix [72] and PANE [73] whereas some controllers support multiple APIs [70, 74].

Onix [72] is a distributed control architecture using multiple physical servers and each server running multiple Onix instances. It provides a general API for scalable application development. The Onix controller provide applications access to data plane devices for reading and writing network state information. Onix uses data centric approach in which applications retrieve consistent network state information from a data structure termed as Network Information Base (NIB). This data structure is divided among multiple controller instances that hold responsibility for the subset of NIB. Strong consistency is ensured using reliable mechanism of replication and transactional database. ONIX also supports distributed hash table (DHT) to provide general APIs that guarantee weak consistency of the network view**.**

PANE [73] architecture also presents a high-level API between applications and control plane. This interface allows end host applications to request network resources like bandwidth reservation, path selection, etc. The PANE control framework uses a verification engine to avoid starvation and conflicts among competing application requests.

One of the most popular open source SDN Controller project is OpenDaylight (ODL) [70], driven by Linux Foundation and supported by research community and industry. The main architectural feature of ODL is Model-Driven Service Abstraction Layer (MD-SAL) based on YANG models [75], which allows simple and flexible integration of network services requested by the application layer via northbound APIs (supporting RESTful interface and OSGi [76] Framework). The YANG interface provides a set of REST APIs by which application developers can configure and retrieve network state information. YANG data models provide inventory information of whole network and collect network statistics information which enables management applications to manage overall network traffic. The distributed controllers in ODL provide a logically centralized view to management plane by using Akka framework [77] and RAFT consensus algorithm [78].

ONOS [74] supported by ON-Lab is a distributed control architecture for scale-out performance and fault tolerance. It uses multiple instances of floodlight controller, each handling subset of OpenFlow switches. On top of the controller it provides wide range of templates for application development. Applications interact with the control plane using java, REST APIs and OSGi bundles. The applications use Cassandra's consistent data store which presents network view as a graph and is used to retrieve and modify network state information. Consistency among application's concurrent access to data store is ensured by Titan's transactional semantics. In ONOS, the distributed controller instances use gRPC [79] for communication among distributed applications.

4.2 Intent-based APIs

Intent-based APIs provide a flexible approach which allows applications in management plane to specify policy-based directives, termed as intents. After receiving intents from management plane, the control plane transforms these directives into necessary forwarding rules which are communicated to the data plane devices. Therefore, with the help of intents, applications can specify the necessary requirements without bothering about how it will be executed by the control plane.

According to authors in [80, 81], the intent-based networking (IBN) involves translation of high-level policies, automated implementation of such policies, awareness of entire network state and assurance of desired state. The overall functioning of intent framework can be visualized as shown in figure 6. The intent service provides necessary application support for defining intents. The intent compilers transform application specific intents into forwarding rules. The intent installers install necessary flow rules into the data plane devices. The two distributed controllers which support intents are ODL [70] and ONOS [74].

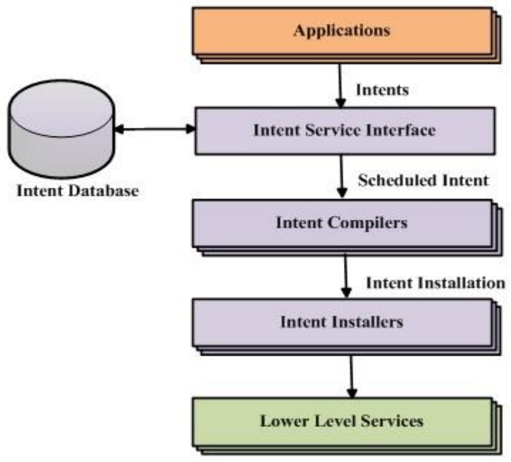

Fig. 6. General Intent Architecture

The ODL control framework is working on Network Intent Composition (NIC) [82, 83], an intent support project which enables external applications to give directives to the ODL core modules. In NIC architecture, the core component is intent model which specifies details of preferred network state. The NIC implementation along with ODL controller transforms user's desired state into necessary flow rules and configuration messages which change the overall state of network resources. The component that performs this transformation is termed as renderer. Various versions of ODL support wide range of renderers which include OpenFlow renderer, Network Modeling renderer (NEMO), group-based policy renderer and virtual tenant network renderer.

There are various components that support the NIC capability in ODL and functioning of renderers. To have a close insight of renderer operation, the architecture of NEMO model [84] is shown in figure 7. The physical network manager obtains and maintains data about the network infrastructure in the MD-SAL data store. This information is used by the user manager module and various other renderers. The object manager sub-module manages the node, flow and connection objects. The behavior of these objects is manipulated as per the policies or directives received from the management plane applications which very often results in their state change. However, such state changes should satisfy the constraints stated by the result monitor module. To get the desired network state, the object manager should notify changes of all objects to the result monitor module and the operation resolver module.

The object manager enables network operators to design virtual networks over the physical network. With the help of object manager's sub-modules (node mapper and connection mapper) and renderers, virtual networks are automatically mapped onto the physical network. The operation resolver sub-module is responsible for monitoring the behavior of various network objects and helps in resolving issues in case of conflicts which may arise when some operations try to affect the same object.

Likewise, the Intent framework [85] in ONOS distributed controller allows management applications to state their requirements in terms of network policies. The policy-based directives or intents in ONOS are identified by

Application_id and Intent_id. The former represents the application that initiated the intent whereas the later represents a particular directive. In ONOS's Intent management framework, when an application submits the intent, it first goes through a compilation phase. The compilation phase checks the necessary requirements and communicates such requirements to the installation phase. While compiling intents, if an application has requested for an unavailable object, the compiler tries to find an alternate, but if an alternative is not available then an error is reported. The installation phase transforms the network state into application desired state. The intent store provides the necessary support during compilation and installation of intents and helps in verifying the network state after intent installation.

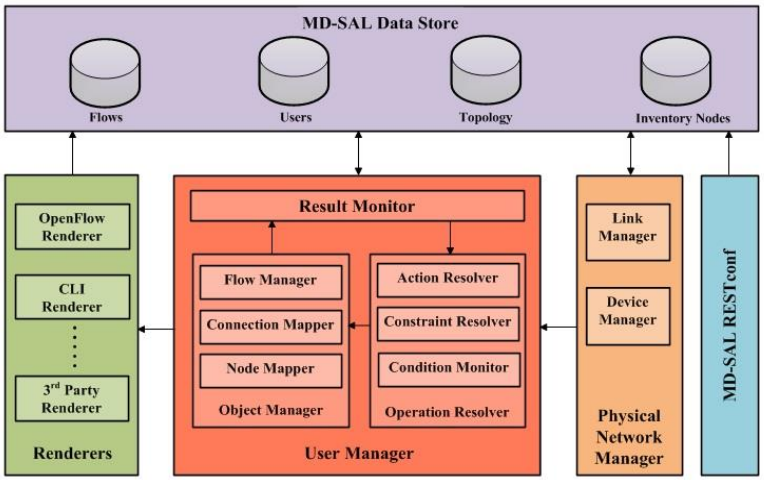

Fig. 7. NEMO architecture

Apart from ODL [70] and ONOS [74], the authors in [86] have proposed a general three tier architecture for intent based NBIs. These tiers include business logic tier, database tier and presentation tier. To have flexibility in network management, these tiers work independently and if there is a change in one tier, it does not affect the other tiers. The database tier stores application states and network information. The business tier handles service creation and configuration. A service registry stores the existing services and new atomic services which can be defined in business tier. The existing and new services are integrated by service integration element. The presentation tier takes service input either with the help of CLI, programming interface or REST API. Further, authors have used domain driven design in [86], wherein requirements are decomposed into sub-problems and solution is determined for each sub-problem.

OSDF [87] framework has been extended in [88] to provide intent based approach for policy definition (covering multiple domains), QoS provisioning and a warning mechanism to register user's unauthorized attempts. The main components of this framework include: i. *High Level Network Operation Services (HLNOS):* provides flow rule, topology, region and configuration services to implement high level network policies. It provides inter and intra-site routing and QoS provisioning services. *ii. Policy Store Module (PSM):* is used to store and retrieve application-based policies dynamically at runtime. *iii. Policy Parser Module (PPM)*: analyzes policies and incoming flows to generate forwarding flow rules. It uses services of following sub-modules: network region parser, traffic selector builder and path selection service. *iv. Policy Conflict Management Module (PCMM):* provides two types of services namely path conflict detection service and path conflict resolution service.

Intent based approach for NBI is a good initiative which provides flexibility, application portability and simplifies application development. However, only few SDN controllers (ODL [70] and ONOS [74]) support intents. The communication protocols used at SBI and NBI in various SDN controllers are summarized in table 4.

4.3 Goal-oriented Approaches

The major hindrance in application development in SDN is a gap between application developers, control plane designers and data plane device vendors. Consequent to it, the applications are mostly locked to the control platform for which they are developed, resulting in portability and reusability issues. Further to handle diverse data plane devices there should be provisions both at NBI and SBI to abstract the architectural differences. To address these issues, numerous approaches have been proposed by researchers which use different mechanisms and provide diverse level of abstractions. Such mechanisms and abstractions simplify and provide flexible interface for transforming business objectives into appropriate flow rules and network configurations. In this section, we have identified and discussed various such proposals.

Table 4. Communication mechanisms used at NBI and SBI in diverse SDN Controllers

Controller	Programming Language	Architecture	Southbound Interface	Northbound Interface	Partner
NOX [60]	C++	Centralized	OF 1.0	ad-hoc	Nicira Networks, USA, 2008
POX [61]	Python	Centralized	OF 1.0	ad-hoc	Nicira Networks, USA, 2013
Trema [62]	Ruby and C	Centralized	OF 1.3, 1.0	ad-hoc	NEC, 2013
ParaFlow [63]	C++	Centralized	OF 1.0, 1.3	Abstract APIs	- , 2017
Rosemary [65]	C	Centralized	OF 1.0, 1.3	ad-hoc	-, 2014
Floodlight [67]	Java	Centralized	OF 1.0 -1.5	REST API	Big Switch Networks, USA, 2012
Ryu [68]	Python	Centralized	OF 1.0-1.5, Netconf., OF-config	REST API	Nippo Telegraph and Telephone Corporation, Japan 2013
MUL [69]	C	Centralized	OF 1.4, 1.3, 1.0, OVSDB, OF-config.	REST API	Kulcloud , 2012
ODL [70]	Java	Flat, Distributed	OF 1.1-1.5, NETConf, OVSDB, PCEP, etc	REST API, OSGi Framework and Intent Support	Linux Foundation, USA, 2013
Onix [72]	Python, C	Flat, Distributed	OF 1.0-1.5	NVP –NB API	Nicira Networks, Google, NEC ICSI & UC Berkeley, 2010
PANE [73]	Java	Hierarchical, Distributed	OF 1.0	PANE API	Brown University, 2013
ONOS [74]	Java	Flat, Distributed	OF1.0, 1.3, NETCONF	REST API, Neutron, Intent Support	ON.LAB, AT&T, Ciena, Cisco, Ericsson, Fujitsu, Huawei, Intel, NEC, Nsf. Ntt Communication, Sk. Telecom, 2014

A. *NOSIX*

NOSIX [89] defines a low-level portable application interface with which diverse data plane devices can be handled. Although, NOSIX cannot be considered as a general-purpose NBI as it only provides a simple layer of abstraction for applications in terms of Virtual Flow Tables (VFT). These VFTs are used by applications to define flow rules without considering the low-level details of notification to the switches. A VFT pipeline can be predefined by the applications and then these virtual tables are installed into the data plane devices by the switch drivers. Such drivers can be placed either on the controller side or may be associated with the switch. With the help of this abstraction layer, NOSIX gives southbound APIs a look of device drivers. The VFTs enables applications to perform dynamic reconfiguration of switches and improves efficiency as all the rules in VFTs need not to be in the physical flow tables. Hence, NOSIX improves portability and efficiency by enabling applications to express their expectations into VFTs and at the same time rely on vendors to build drivers to transform them into switch specific configurations.

B. *High-level NBI*

In [90], the authors have proposed a high-level NBI that provides abstractions to transform high-level requirements (business objectives) defined by the network administrator into network configurations. It defines two layers at NBI: service mapper and management function mapper. The former is responsible for mapping the high-level requirements to the relevant service presented by the provider whereas the later selects a particular set of functions that need to be invoked to satisfy the technical requirements of desired service. In the proposed system, the network environment is abstracted as resources (e.g., link, switch, server etc.) and commodities (traffic, virtual machine, etc.) to the application programmer. The upper layer or service mapper receives high level application requirements and updates the technical specifications as per the received input. The lower layer or management function mapper executes the specific control software modules that meet the technical requirements.

Additionally, in [90], the control software or controller is represented as a network management function orchestration. To implement the technical requirements, the management functions consider the values of long-lived parameters such as infrastructure attributes (e.g., link, storage, etc.), as well as short-lived parameters, representing the current context dynamics (e.g., resource utilization, capacity, demand, etc.). Such parameters are defined based on an abstract view of the overall network infrastructure and are stored in a network repository. Based on the calculated outcome, the management functions perform the necessary configuration changes. The authors have implemented the prototype in java and have used mininet [91] for network emulation.

C. *SFNet*

SFNet [92] provides an elementary abstraction layer over the SDN controller which allows applications to interact with underlying network devices using a high-level API. It provides high-level primitives by which network applications can verify network status and request resource reservation. The applications use JSON messages to determine the network status and can reserve bandwidth for traffic flows. Based on the network status and available bandwidth, access is granted or denied to the applications. Further, with the help of bandwidth reservation feature, it can avoid network congestion; prioritize real-time traffic like VoIP or video on demand. The proposed approach is beneficial for delay sensitive traffic and supports multicast sessions between network applications.

D. tinyNBI

tinyNBI [93] provides language independent low-level interface to address portability issue of OpenFlow versions. The primary objective of tinyNBI is to provide abstractions to application developers to address variations in OpenFlow versions and switch capabilities. Each such abstraction involves three components which are capabilities, configuration and statistics. Capabilities are a non-amendable state that defines abstraction's range of behaviors which could be even a single behavior. Configuration is the data that changes the behavior of abstraction with the help of interface. Statistics represents read-only data that gives depiction of how an abstraction has performed. More specifically, abstractions represent features of OpenFlow versions like meter table, group table etc. In some OpenFlow versions all abstractions are not present. Such abstractions are handled either by seamless emulation or switch offloading or even with an error signal. Emulation defines abstraction with limited abilities. Switch offloading offloads missing switch capabilities to the controller. However, in certain cases, it is impractical to provide missing behavior which is represented by an error signal. TinyNBI basically is a data model which does not provide information like switching or routing, network topology etc. Instead, it gives an abstraction to programmers by which semantics of OpenFlow versions are handled effortlessly.

E. REST-like NBI

In [94], the authors argue that the current implementations and designs of NBI have numerous limitations, particularly lack of security features. They have proposed a REST-like interface which first registers SDN applications and then such applications can retrieve network resource information like traffic statistics, topology data or flow information. They have proposed a trust manager by which only authenticated applications can utilize the services of NBI. All connection requests from management applications first go through mutual authentication with the controller using transport Layer Security (TLS) and X.509 certificates. After successful authentication, the applications can issue directives to the controller service and receive information about various networking events by registering themselves for appropriate event listeners. Further, the controller keeps track of applications state to ensure proper authorization. Such controller services are provided using Java APIs for RESTful Services (JAX-RS) with reference implementation Jersey [95]. With the help of JAX-RS, it becomes trivial to export internal controller objects like switch, topology or host information. Further, Jersey provides an extension that supports Server-Sent Events [96], which is used to implement the event system.

Table 5. Summary of Goal-oriented APIs for NBI

NBI	Mechanism	Main Highlights
NOSIX [89]	Virtual Flow Tables (VFTs)	Provides portable low-level application interface which improves portability and efficiency by enabling applications to express their expectations in VFTs
High Level NBI [90]	Two layers at NBI: Service mapper and management functions mapper	The top layer or service mapper receives high level application requirements and updates the technical specifications as per the received input whereas the lower layer or management functions mapper executes specific control software modules that meet the technical requirements.
SFNet [92]	Abstraction Layer with JSON messaging service	It provides high-level primitives to network applications with which they can verify network status and request resource reservation using JSON messages
tinyNBI [93]	Abstractions to express capabilities, configuration and statistics	It provides abstractions for application development to address variations in OpenFlow versions and switch capabilities
REST-like [94]	TLS and X.509 Certificates	A secure REST-like interface which enables SDN applications to register first and then such applications can view network resources such as traffic statistics, topology data or flow information.

Summary: The various proposals proposed for NBI are mostly tied to the controller platforms [60-65], with majority of controllers supporting REST APIs [67-70] and only few support intent-based APIs [70, 74]. Although intent-based NBIs simplifies application development, however it needs to be further enhanced and extended to diverse controller platforms. Apart from it, various goal-oriented approaches have tried to achieve a specific control objective. With such diverse approaches, it is difficult to achieve application portability, reusability and interoperability among diverse SDN controllers. An extensive, robust and mature solution is the need of the hour at the NBI.

5. Future Research Perspectives

The efficient utilization of network resources is determined by the effective communication between SDN planes and selection of algorithms in the control and management plane. The communication protocols at SDN interfaces play a vital role in achieving network wide view at a logically centralized SDN controller and to streamline various network operations like flow management, load balancing, dynamic policy management, etc. In this section, we have highlighted the challenges associated with existing approaches and presented future research perspectives in south and northbound APIs. We have highlighted the importance of secure communication mechanisms required at south and northbound interface in order to achieve effective network monitoring and virtualization in next generation networks. Further, we have proposed certain extensions and abstractions (shown in figure 8) which need to be combined with existing prominent solutions for SBI/NBI and are emphasized and expounded here in this section.

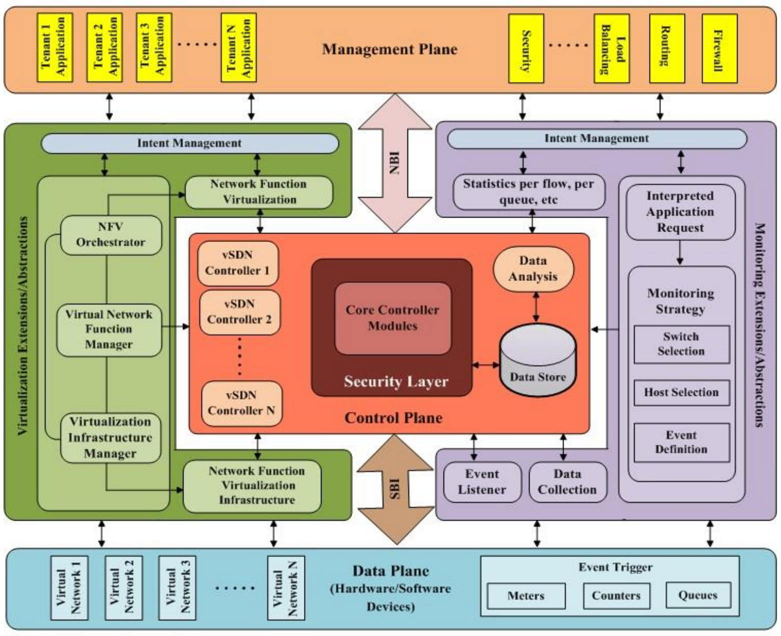

Fig. 8. Proposed extensions and abstractions for NBI and SBI to address monitoring, virtualization and security requirements.

5.1 Monitoring Support at NBI and SBI

Network monitoring is a fundamental element in network management [97]. Management applications require perfect and appropriate statistics of network state at diverse aggregation levels. The major design challenge is to reduce the monitoring overhead which may otherwise have an impact on normal network traffic flow [98]. The monitoring frameworks proposed for pure SDNs include OpenSketch [99], Payless [100], OpenNetMon [101], FlowCover [102] and Probe-SDN [103]. Most of these frameworks are basically modules integrated with SDN controllers to collect monitoring data using either active polling or sampling.

OpenSketch [99] employs sketch based streaming algorithms for performing measurements on traffic flow streams in the data plane using three stage processing pipeline, comprising of hashing, filtering and finally counting. Although, OpenSketch requires limited memory and provides comparable accuracy with respect to counter-based mechanisms, but lacks complex data structures and measurement algorithms. On the other hand, Payless [100] is a monitoring framework built upon floodlight controller which involves number of modules for translating application specific requests regarding flow statistics into flowstat response. It incurs petite overhead and provides accurate monitoring data in a timely manner. However, the major limitation in PayLess is the assumption that the effective and promising monitoring approaches can be developed only using OpenFlow.

OpenNetMon [101] and Probe-SDN [103] focus on tradeoff between accuracy and overhead. OpenNetMon mainly focuses on traffic engineering whereas Probe-SDN on evaluation of bandwidth utilization. However, such solutions are not universal and are specific to a particular application scenario. FlowCover [102] exploits SDN's network visibility and single point of network control to achieve high-accuracy monitoring scheme at a low-cost. FlowCover provides a dynamic approach to select target switches for flows rather than using a per-flow basis analysis. All these studies have mainly focused on communication between data and control plane, and usually involve the following four functions: statistics collection, aggregation, data analysis and storage of monitoring data. However, most of these approaches fail to innovate the ways of flow statistics collection from switches and does not address various application requirements. Nowadays, the application domain to be monitored is continuously changing and growing. With emerging fine-grained monitoring requirements, the monitoring mechanism provided by OpenFlow is very limited.

Keeping in view the aforesaid requirements, we believe that the communication mechanisms used at SBI and NBI should be extended to provide necessary abstractions and extensions to support an effective monitoring mechanism in next generation networks. As shown on the right side of figure 8, at NBI, there should be provisions to send monitoring related application intents to the control plane and at the same time the management applications should be able to retrieve monitoring data in varied forms. The network control logic in SDN controller implements monitoring strategy and receives monitoring data from the data plane using SBI. The two most common monitoring strategies used in the aforesaid monitoring frameworks are switch selection and end-host selection. In switch selection, different switch selection algorithms are used to achieve target switch set whereas in end host selection, the two appropriate end hosts are chosen to reduce the overhead incurred in active measurement. However, we believe that there should be provisions to define network events and event triggers in data plane to reduce the bi-directional communication overhead, as shown in figure 8. The same is being debated by the researcher community using programmable data planes [104]. If such event triggers are deployed in the data plane devices, the collection element of each event trigger can constantly gather the monitoring data. On the basis of collected data, the analysis element can calculate the current network state in a timely manner. OpenFlow need to be extended or novel solutions need to be designed to enable dynamic event management in the data plane devices.

5.2 Provisions for Network Virtualization

Network Function Virtualization (NFV) and SDN are two different paradigms with former aimed to decouple network functions from specialized hardware while as the later focuses on segregation of network control logic from packet handling. NFV and SDN are highly complementary to each other, as both advocate creativity, competitiveness, virtualization and automation to realize their respective goals. The common objective of NFV and SDN is to promote standard network hardware and open software. Hence, combining them into a single networking solution may yield greater value [105].

The researchers have proposed multiple approaches like FlowVisor [106], OpenVirteX [107] at SBI and FlowN [108], NVP [109], libNetVir[110] at NBI, which provide different extensions and abstractions. However, majority of these solutions address a particular issue like addressing or topology virtualization at SBI and separation of tenant applications or control logic at NBI. To meet the demands of NFV, we believe that both SBI and NBI along with SDN controller design need to be enhanced. Since the focus of manuscript is on NBI and SBI, so here we have only elaborated the issues associated with these two interfaces.

At SBI, OpenFlow is the most widely used solution which mainly targets L2-L4 packet handling. It lacks upper layer protocol support (L5-L7) and application-oriented flow control. The authors in [111] have evaluated OpenFlow in terms of data classification, QoS, tunneling, etc, and have concluded that in order to support NFV, OpenFlow needs to be extended. Further, for various tasks of virtualization the other solutions like NetConf [58] need to be considered at this interface. On the other hand, consequent to the absence of standard NBI and diversity in data models used in different controller platforms, achieving uniform virtualization support across diverse control platforms is also very difficult.

Keeping in view the said issues, the communication mechanisms at SBI and NBI need to be enhanced and extended to enable network operators to define multiple virtual network slices over the same physical network and facilitate implementation of various tenant applications on different virtual controllers as shown in figure 8. To complement NFV with SDN, we believe communication mechanisms at SBI and NBI need to be extended and enhanced to provide a multi-level support for NFV.

5.3 Open Issues in Existing Approaches

OpenFlow [32, 33] and ForCES [54] are two renowned solutions for SBI. Although, ForCES was much before the OpenFlow, but OpenFlow has gone through a brisk evolution in the recent past. However, still there are certain limitations in OpenFlow like lack of support for higher layer packet processing and wireless LAN (WLAN) protocol header fields. Currently, various versions of OpenFlow support L2-L5 packet processing, however to perform fine-grained flow diversion and deep packet inspection, there should be support for upper layer packet headers as well. Further, latest OpenFlow version 1.5 supports 41 matching fields which require more space and processing in OpenFlow devices. To reduce this processing and space burden for OpenFlow complaint devices, solutions like POF [48] can be integrated with OpenFlow to enable network operators to define fields as per the requirement in data plane. The other possibility is to use programmable data plane which enable users to define packet processing logic flexibly as per the demand, therefore enables adaptability and scalability in packet processing [112, 113].

ForCES is also a mature solution which provides extensibility features without changing the existing architecture. It supports rich set of features which are still non-existent in OpenFlow or other SBI protocols discussed in section 3. A possible solution could be to combine features of OpenFlow and ForCES into a single sophisticated solution; however, it may result in increased overhead and complex implementation. Further, research contribution is required to revisit and evaluate both solutions with the aim to formulate a solution which can be tuned to specific parameters depending on the type of network.

On the other hand, at NBI, consequent to the diverse control platforms (distributed, centralized and hybrid controllers), different data models used in controllers, and absence of standard NBI hampers interoperability and application portability in SDNs. Therefore, to hide the complexity of control logic and provide necessary services and abstractions to applications, a flexible and extensible NBI is required. Such a standard NBI is important to provide virtualization and other services at NBI. Moreover, Intent based networking has received some traction from the research community, however, it requires further investigation and research contribution to integrate and implement it effectively with different control platforms.

5.4 Security and Privacy Issues

The effective and efficient forwarding in data plane depends on the flow rules communicated by the control plane with the help of communication protocols used at SBI. OpenFlow is the de-facto standard at this interface but transport layer security (TLS) is not mandatory, making management of network infrastructure susceptible to security threats. Consequent to the absence of TLS based deployments, numerous vulnerabilities may be hidden. Experimental evaluation on large scale test-beds is required to divulge more information on OpenFlow security. Additionally, TLS is itself vulnerable to various security attacks, mainly man-in-the-middle attack. Other alternatives like IPSec or Secure Shell should be considered to secure information exchange at this interface.

Authors in [114] have presented security analysis of OpenFlow using security threat modeling technique STRIDE [115]. As per their observation, information disclosure and Denial of Service (DoS) attacks can be easily performed against OpenFlow complaint devices. Information disclosure attacks expose the state and services of the network. Proactive strategies and rate limiting techniques like flow aggregation and meters can be used to prevent such attacks. Such strategies and techniques should be incorporated into a security framework which can continuously monitor packet processing in the data plane and resource utilization in the control plane for detection and mitigation of possible security threats.

On the other hand, the application of centralized SDN control in diverse domains can be realized by using different third-party applications in the management plane which use services of control plane through NBI. However, neither the security requirements nor the communication protocol have been standardized for this interface. Consequent to it, the application designers have to consider necessary security measures in their applications as per the control platform, to prevent unauthorized access of control plane services. Such measures make application development cumbersome and applications remain confined to a particular controller.

Within a controller, there should be measures like process containment, application permission structure and resource utilization monitoring so that the attacks like spoofing, tampering, denial of service (DoS), privileges elevation can be mitigated. If there are no security measures at NBI, an unauthorized application can gain access to the controller services, with the result, it can corrupt the entire network information which can be catastrophic for an organization.

Currently, only two control frameworks OpenDayLight [70] and ONOS [74] provide security mechanisms for unauthorized access and tampering [26]. OpenDayLight supports AAA project [116] which uses token-based security mechanism for applications to access network resources. However, this approach authenticates a user rather than the application. To address this issue, authors in [117] have used OAuth 2.0 [118] framework which authenticates both application and the user. In the proposed framework, the user/application is registered first with an authentication server. After successful authentication, the user/application is granted access to the network services. On the other hand, ONOS employs HTTPS at northbound interface to prevent tempering and information disclosure threats. Authorized access and fine-grained control to internal data-structures and libraries are provided in Secure-Mode ONOS (SM-ONOS). In this mode, it uses strict access control measures and security audit service to prevent repudiation and elevation of privilege threats.

Additionally, in SDN complemented network virtualization, securing different network slices and virtual SDN controllers is an intricate task as there may be different security requirements for various network slices and the resources are shared among slices. These network slices must be protected from each other and security measures should be considered while designing virtual SDN controllers. We believe that a security wrapper is required around the SDN controller to secure it both horizontally (east/westbound interfaces) and vertically (north/southbound interfaces) as shown in figure 8.

6. Conclusion

SDN enables network innovations and simplifies network management, however its interfaces demand an immediate attention from academia and industry. In this manuscript, we have presented a systematic and comprehensive survey of various protocols/proposals addressing north and southbound SDN interfaces. We have classified these proposals into various categories based on their functionality, dependence and properties. We have analyzed these protocols/standards and have identified numerous open issues and prospective trends which requires immediate attention from the research community.

OpenFlow is the most renowned de-facto industry standard for SBI which promotes innovations and advances in forwarding plane and simplifies management and monitoring of forwarding devices. Many approaches have extended or modified OpenFlow to apply it in different domains or address its limitations. However, there are still many open issues and challenges associated with this prominent standard which we have highlighted along with the probable solutions. On the other hand, consequent to the absence of standard NBI, various approaches have been used at this interface which we have classified into following three categories: in-built controller specific, intent based and goal-oriented. We have analyzed all such approaches and have highlighted pros and cons associated with each proposal. With these diverse approaches, it is very difficult to achieve application portability, reusability and interoperability among diverse SDN controllers. We believe that an extensive, robust and mature solution is required at northbound interface to address such issues.

In addition to it, we have proposed extensions and abstractions required at these two interfaces to support effective monitoring and virtualization in next generation networks. The proposed extensions/abstractions not only simplify the overall network management but are also useful in realizing benefits from this radical network architecture in diverse domains. Further, we have emphasized the importance of securing information exchange at these interfaces in order to prevent malicious applications/devices to disrupt the overall network functioning. We believe that the existing SBI and NBI protocols/proposals are still in infancy stage and require further extensions and enhancements to enable wide range adoption of agile and programmable SDN architecture in diverse domains.

References

[1] Internet Usage Statistics. [Online] Available: https://www.internetworldstats.com/stats.htm

[2] Cisco Annual Report. [Online] Available: https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report/infographic-c82-741491.html

[3] Zilberman, N., Watts, P. M., Rotsos, C., and Moore, A. W. (July 2015), Reconfigurable Network Systems and Software-Defined Networking. IEEE (pp. 1102–1124), 103(7).

[4] Colville, R. J., and Spafford, G. (2010). Configuration Management for Virtual and Cloud Infrastructures. Gartner Inclusive, [Online] Available: http://www.gartner.com/id=1458131.

[5] Release, P. Hacking Habits: The survey cites misconfigured networks as the main cause of breaches. Tufin Technologies, [Online] Available: http://www.tufin.com/about-us/news-and-media/ press-releases/august-31.

[6] Ghodsi, Shenker, S., Koponen, T., Singla, A., Raghavan, B., and Wilcox, J. (2011). Intelligent Design Enables Architectural Evolution. In Proceedings of the tenth ACM Workshop on Hot Topics in Networks, New York, USA.

[7] Raghavan, B., Casado, M., Koponen, T., Ratnasamy, S., Ghodsi, A., and Shenker, S. (2012). Software-Defined Internet Architecture: Decoupling Architecture from Infrastructure. In the Proceedings of the Eleventh ACM Workshop on Hot Topics in Networks, 43–48.

[8] ONF White Paper. (2012, April). Software Defined Networking: The New Norm for Networks.

[9] Sezer, S., Scott-Hayward, S., Chouhan, P.K., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M., and Rao, N. (2013, July). Are we ready for SDN? Implementation challenges for software-defined networks. IEEE Communications Magazine, 51(7).

[10] Cox, J. H., Chung, J., Donovan, S., Ivey, J., Clark, R. J., Riley, G., Owen, H. L. (2017). Advancing Softwared Defined Networks: A survey. IEEE Access, 5.

[11] Wang, H., Xu, H., Huang, L.and Wang, J. (2018, May). Load-Balancing Routing in Software Defined Networks with Multiple Controllers. Computer Networks, 141.

[12] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J., (2008, March). OpenFlow: enabling innovation in campus networks", SIGCOMM Computer Communication Review (pp. 69–74), 38(2).

[13] Kreutz D., Ramos, F. M. V., Verissimo, P., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S., (2015, January). Software-Defined Networking: A Comprehensive Survey. IEEE, 103(1), DOI: 10.1109/JPROC.2014.2371999.

[14] Feamster, N., Rexford, J. and Zegura, E., (2014, April). The Road to SDN: An Intellectual History of Programmable Networks. In ACM SIGCOMM Computer Communiation Review (pp. 87-98), 44(2).

[15] Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K., and Turletti, T. (2014). A Survey of Software Defined Networking: Past, Present and Future of Programmable Networks. IEEE Comm. Survey Tutorials (pp. 1617–1634), 16(3).

[16] Jarraya, Y., Madi, T., and Debbabi, M., (2014). A Survey and a Layered Taxonomy of Software Defined Networking. IEEE Commun. Surveys and Tutorials (pp. 1955–1980), 16(4).

[17] Hu, F., Hao, Q. and Bao, K., (2014). A Survey on Software Defined Network and Openflow: From Concept to Implementation. IEEE Comm. Surveys & Tutorials (pp. 2181–2206), 16(4).

[18] Jarchel, M., Zinner, T., Hossfeld, T., Tran-Gia, P., and Keller, W. (2014). Interfaces, Attributed and Use Cases: A Compass for SDN. IEEE Communication Magazine.

[19] Bera, S., Misra, S., and Vasilakos, A. V. (2017, December). Software Defined Networking for Internet of Things: A Survey. IEEE Internet of Things Journal (pp. 1994–2008), 4(6).

[20] Gong, Y., Huang, W., Wang, W., and Lei, Y. (2015, December). A Survey on Software Defined Networking and Its Applications. Frontiers of Computer Science (pp. 827–845), 9(6).

[21] Modieginyane, K. M., Letswamotse, B. B., Malekian, R., and Abu-Mahfouz, A. M. (2018 February). Software Defined Wireless Sensor Networks Application Opportunities for Efficient Network Management: A Survey. Computers & Electrical Engineering (pp. 274–287), 66.

[22] Hakiri A., Gokhale A., Berthou P., Schmidt D. C., and Gayraud T. (2014, December). Software Defined Networking: Challenges and Research Opportunities for Future Internet. Computer Networks (pp. 453– 471), 75(Part A).

[23] Yu, Y., Li, X., Leng, X., Song, L., Bu, K., Chen, Y., Yang, J., Zhang, L., Cheng, K., and Xiao, X. (2018, September). Fault Management in Software Defined Networking: A Survey. IEEE Communications Surveys & Tutorials (pp. 349 – 392), 21(1).

[24] Fonseca, P., and Mota, E., (2017). A Survey on Fault Management in Software Defined Networks. IEEE Communications Surveys & Tutorials (pp. 2284–2321), 19(4).

[25] Mendiola, A., Astorga, J., Jacob, E., and Higuero, M. (2017). A Survey on the Contributions of Software Defined Networking to Traffic Engineering. IEEE Communications Surveys & Tutorials (pp. 918– 953), 19(2).
[26] Akyilidiz, I. F., Lee, A., Wang, P., Luo, M., and Chou ,W. (2014). A Roadmap for Traffic Engineering in SDN-OpenFlow Networks. Computer Networks (pp. 1-30),71.
[27] Ahmad, S., Mir, A. H. (2020). Scalability, Consistency, Reliability and Security in SDN Controllers: A Survey of Diverse SDN Controllers. Journal of Network and Systems Management, Springer, 29(9) ,doi.org/10.1007/s10922-020-09575-4.
[28] Bannour, F., Souihi, S., and Mellouk, A. (2018). Distributed SDN Control: Survey, Taxonomy, and Challenges. IEEE Communications Surveys & Tutorials (pp. 333–354), 20(1).
[29] Hayward, S. S., Natarajan, S., and Sezer. S. (2016). A Survey of Security in Software Defined Networks. IEEE Communications Surveys & Tutorials (pp. 623– 654), 18(1).
[30] Dargahi, T., Caponi, A., Ambrosin, M., Bianchi, G., and Conti, M. (2017). A Survey on the Security of Stateful SDN Data Planes. IEEE Communications Surveys & Tutorials (pp. 1701–1725), 19(3).
[31] Yan, Q., Yu, F. R., Gong, Q., and Li, J., (2016). Software-Defined Networking SDN and Distributed Denial Of Service (DDOS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues and Challenges. IEEE Communications Surveys & Tutorials (pp. 602– 622). 18(1).
[32] ONF-Open Networking Foundation. [Online] Available: https://www.opennetworking.org/
[33] Limoncelli, T. A. (2012, August). Openflow: a radical new idea in networking. ACM Communication (pp. 42-47), 55(8).
[34] Open Networking Foundation North Bound Interface Working Group (NBI-WG) Charter; VERSION: V 1.1 Available: https://www.opennetworking.org/images/ stories/downloads/working-groups/charter-nbi.pdf
[35] Voellmy, A., Wang, J. C. (2012). Scalable Software Defined Network Controllers. In Proceedings of ACM SIGCOMM conference on Applications, Technologies, Architectures and Protocols for Computer Communication (pp. 289-290).
[36] OpenFlow Switch Specifications Version 1.0. [Online] Available: https://www.opennetworking.org/wp-content/uploads/2013/04/openflow- spec- v1.0.0.pdf
[37] OpenFlow Switch Specifications Version 1.1. [Online] Available: https://3vf60mmveq1g8vzn48q2o71a- wpengine.netdna-ssl.com/wp- content/uploads/2014/10/openflow- spec- v1.1.0.pdf
[38] OpenFlow Switch Specifications Version 1.2. [Online] Available: https://www.opennetworking.org/ images / stories / downloads / sdn - resources / onf - specifications / openflow / openflow- spec- v1.2.pdf
[39] OpenFlow Switch Specifications Version 1.3. [Online] Available: https://www.opennetworking.org/ images / stories / downloads / sdn - resources / onf - specifications / openflow / openflow- spec- v1.3.0.pdf
[40] OpenFlow Switch Specifications Version 1.4. [Online] Available: https://www.opennetworking.org/ images / stories / downloads / sdn - resources / onf - specifications / openflow / openflow- spec- v1.4.0.pdf
[41] OpenFlow Switch Specifications Version 1.5. [Online] Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow- switch- v1.5.1.pdf
[42] Forwarding Abstraction Working Group. [Online] Available: https://www.opennetworking.org/images/stories/ downloads/working- groups/charter- forwarding- abstractions.pdf
[43] Belter, B., Binczewski, A., Dombek, K., Juszczyk, A., Ogrodowczyk, L., Parniewicz, D., Stroiski, M., and Olszewski, I. (2014, Sept.). Programmable abstraction of datapath. In third European Workshop on Software Defined Networks (pp. 7–12).
[44] Parniewicz, D., Corin, R. D., Ogrodowczyk, L., Fard, M. R., Matias, J., Gerola, M., Fuentes, V., Toseef, U., Zaalouk, A., Belter, B., Jacob, E., and Pentikousis, K. (2014). Design and implementation of an openflow hardware abstraction layer. In Proceedings of ACM SIGCOMM Workshop on Distributed Cloud Computing (pp. 71–76), New York, USA.
[45] Casey, D. J., and Mullins, B. E. (2015). SDN Shim: Controlling Legacy Devices. In the Proceedings of IEEE Conference on Local Computer Networks (pp. 169–172).
[46] Jin, C., Lumezanu, C., Xu, Q., Zhang, Z.-L., and Jiang, G. (2015). Telekinesis: Controlling Legacy Switch Routing with Openflow in Hybrid Networks. In Proceedings of ACM SIGCOMM Symposium on Software Defined Networking Research (pp. 1–7), ACM.
[47] Suarez, R., Rincon, D., Sallent, S. (2015). Extending OpenFlow for SDN-enabled synchronous Ethernet networks. In Proceedings of first IEEE Conference of Network Softwarization (NetSoft).
[48] Song, H. (2013). Protocol oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane. In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (pp. 127–132), USA: ACM.
[49] Bianchi, G., Bonola, M., Capone, A., and Cascone, C. (2014). Openstate: Programming platform-independent stateful openflow applications inside the switch. SIGCOMM Computer Comm. Review (pp. 44–51), 44(2).
[50] Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. (2011, August). Devoflow: Scaling flow management for high- performance networks. SIGCOMM Comput. Commun. Rev. (pp. 254–265), 41(4).
[51] Sune, M., Alvarez, V., Jungel, T., Toseef, U., and Pentikousis, K. (2014). An OpenFlow implementation for network processors. In Third European Workshop on Software Defined Networks.
[52] OpenFlow Management and Configuration Protocol (2014) [Online]. Available: https : / / www. opennetworking . org / images / stories / downloads / sdn - resources / onf- specifications/openflow- config/of- config- 1.2.pdf
[53] Pfaff, B., and Davie, B., (2013, Dec.). The Open vSwitch Database Management Protocol. RFC 7047 (Informational), Internet Engineering Task Force, [Online] Available: http://www.ietf.org/rfc/rfc7047.txt
[54] Haleplidis, E., Salim, J. H., Halpern, J. M., Hares, S., Pentikousis, K., Ogawa, K., Wang, W., Denazis, S., and Koufopavlou, O., (2015). Network programmability with ForCES. IEEE Communications Surveys Tutorials (pp. 1423–1440), 17(3).
[55] Doria, A., Salim, J. H., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and Halpern, J. (2010, Mar.). Forwarding and Control Element Separation (ForCES) Protocol specification. Internet Engineering Task Force. [Online] Available: http://www.ietf.org/rfc/rfc5810.txt.
[56] Hares, S.,. Analysis of Comparisons between OpenFlow and ForCES. [Online]. Available: https://tools.ietf.org/ pdf/draft-hares- forces- vs- openflow- 00.pdf.

[57] Smith, M., Dvorkin, M., Laribi, V., Pandey, V., Gerg, P., and Weidenbacher, N. OpFlex Control Protocol. [Online] Available: https://tools.ietf.org/html/draft- smith- opflex- 03.

[58] Enns, R., Bjorklund, M., Schoenwaelder, J., and Bierman, A. (2011, June). Network Configuration Protocol (NETCONF). [Online] Available: https://www.rfc-editor.org/rfc/pdfrfc/rfc6241.txt.pdf.

[59] Dix, J. (2013, May). Clarifying the Role of Software-Defined Networking Northbound APIs. [Online] Available: http://www.networkworld.com/ news/2013/050213-sherwood-269366.html.

[60] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). NOX: Towards an Operating System for Networks. Computer Communication Review, 38(3).

[61] POX. [Online] Available: http://www.noxrepo.org/pox/about-pox/

[62] TREMA SDN Controller Framework. [Online] Available: https://trema.github.io/trema/

[63] Song, P., Liu, Y., Liu, C., Qian, D. (2017). ParaFlow: Fine-grained Parallel SDN Controller for Large-Scale Networks. Journal of Network and Computer Applications, DOI: http://dx.doi.org/10.1016/j.jnca.2017.03.009

[64] Zhang, S., Shen, Y., Herlich, M., Nguyen, K., Ji, Y., Yamada, S. (2015). Ryuo: Using high level northbound API for control messages in software defined network. In Proceedings of Seventeenth IEEE Asia-Pacific Network Operations and Management Symposium (APNOMS).

[65] Shin, S., Song, Y., Lee, T., Lee, S., Chung, J., Porras, P., Yegneswaran, V., Noh, J., and. Kang, B. B. (2014). Rosemary: A Robust, Secure and High-Performance Network Operating System. in Proceedings of ACM SIGSAC Conference on Computer and Communications Security (pp. 78–89).

[66] Boost. Version 1.61.0. [Online] Available:https:www.boost.org.

[67] Flood light Controller. [Online] Available: https://projectfloodlight.org.

[68] Ryu SDN Framework. [Online] Available: https://osrg.github.io/ryu/.

[69] MUL SDN Contoller. [Online] Available: http://www.openmul.org/.

[70] OpenDaylight: A Linux Foundation Collaborative Project. [Online] Available: https://www.opendaylight.org

[71] Richardson, L., and Ruby, S. (2008). RESTful web services. O'Reilly Media Inc.

[72] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata ,Y., Inoue, H., Hama, T., and Shenker, S. (2010). Onix: a distributed control platform for large-scale production networks. In Proceedings of the Ninth USENIX conference on Operating Systems Design and Implementation, CA, USA USENIX Association.

[73] Ferguson, A. D., Guha, A., Liang, C., Fonseca, R., and Krishnamurthi, S. (2013, Aug.). Participatory networking: An API for application control of SDNs. SIGCOMM Computer Communication Review (pp. 327–338). 43(4).

[74] Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., and Parulkar, G. (2014). ONOS: Towards an Open, Distributed SDN OS. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, New York (pp. 1–6), USA: ACM.

[75] Bjorklund, M. (2016). YANG - A Data Modeling Language for Network Configuration Protocol. RFC-7952.

[76] OSGi Specification. [Online] https://www.osgi.org/developer/what-is-osgi/

[77] Akka Framework. [Online] Available: http://akka.io/

[78] Ongaro, D., and Ousterhout, J. (2014). In Search of an Understandable Consensus Algorithm. In Proceedings of USENIX Annual Technical Conference (pp. 305–320), CA, USA.

[79] What is gRPC? [Online]. Available: https://grpc.io/docs/guides/

[80] Intent-Based Networking: Automating Next-Generation Networks, Heavy Reading Report, [Online]. Available: http://www.heavyreading.com/details.asp?sku_id=3506&skuitem_itemid=1731.

[81] Butler, B. What is intent-based networking. [Online] Available: https://www.networkworld.com/article/3202699/what-is-intent-based-networking.html

[82] Network Intent Composition (NIC) Developer Guide [Online]. Available: https://docs.opendaylight.org/en/stable-fluorine/developer-guide/network-intent-composition-(nic)-developer-guide.html

[83] Rodrigues, Y. OpenDaylight ODL: Network Intent Composition (NIC) - A real Intent-based solution, challenges and next steps Intent Framework. [Online] https://www.serro.com/opendaylight-network-intent- composition- a- real- intent- based- solution-challenges- and- next- steps/

[84] NEMO PROJECT. [Online] https://docs.opendaylight.org/en/stable-fluorine/release-notes/projects/nemo.html

[85] Intent Framework. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Intent+Framework

[86] Pham, M. and Hoang, D. B. (2016, June). SDN applications - the intent-based northbound interface realisation for extended applications. In the Proceedings of IEEE NetSoft Conference and Workshops (NetSoft) (pp. 372–377).

[87] Comer, D., Rastegarnia, A. (2017, Oct.). OSDF: A Framework For Software Defined Network Programming. arXiv:1710.00958v1 [cs.NI].

[88] Comer, D., Rastegarnia, A. (2018, July). OSDF: An Intent-based Software Defined Network Programming Framework. arXiv:1807.02205v1 [cs.NI].

[89] Yu, M., Wundsam, A., and Raju, M. (2014, Apr.). NOSIX: A lightweight portability layer for the SDN OS. SIGCOMM Computer Communication Review (pp. 28–35), 44(2).

[90] Tuncer, D., Charalambides, M., Tangari, G., & Pavlou, G. (2018, November). A Northbound Interface for Software-based Networks. In the proceedings of Forteenth International Conference on Network and Service Management (CNSM) (pp. 5-9).

[91] Lantz, B., Heller, B., and McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. In Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks, New York, USA, ACM.

[92] Yap, K.-K., Huang, T.-Y., Dodson, B., Lam, M. S., and McKeown, N. (2010). Towards software-friendly networks. In Proceedings of the First ACM Asia-pacific Workshop on Workshop on Systems (pp. 49–54), ACM.

[93] Casey, C. J., Sutton, A., and Sprintson, A. (2014). TinyNBI: Distilling an API from essential openflow abstractions. CoRR, vol. abs/1403.6644.

[94] Banse, C., and Rangarajan, S. (August, 2015). A Secure Northbound Interface for SDN Applications. In the proceedings of IEEE international conference on Trustcom/BigDataSE/ISPA.

[95] Jersey. [Online] Available: https://jersey.java.net
[96] Server Sent Event. [Online] Available: http://www.w3.org/TR/eventsource/
[97] Kim, J., Sim, A., Sang, C.S., Kim, I. (2017). An approach to online network monitoring using clustered patterns. In proceedings of the International Conference on Computing, Networking and Communications,Silicon Valley (pp. 656–661), USA.
[98] Shahreza, S. S., Ganjali, Y. (2015). Traffic statistics collection with FleXam. In Proceedings of the ACM Conference on SIGCOMM (pp. 117–118), London, UK.
[99] Yu, M., Jose, L., and Miao, R. (2013). Software Defined Traffic Measurement with Opensketch. In the Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI) (pp. 29–42).
[100] Chowdhury, S. R., Bari, M. F., Ahmed, R., and Boutaba, R. (2014). Payless: A low cost network monitoring framework for software defined networks. In the Proceedings of IEEE Network Operations and Management Symposium (NOMS) (pp. 1–9).
[101] Adrichem, N. L. V., Doerr, C., and Kuipers, F. A. (2014). OpenNetMon: Network Monitoring in Openflow Software Defined Networks. In the Proceedings of IEEE Network Operations and Management Symposium (NOMS) (pp. 1–8).
[102] Su, Z., Wang, T., Xia, Y., Hamdi, M. (2015, December). FlowCover: Low-cost flow monitoring scheme in software defined networks. In Proceedings of the Global Communications Conference (pp. 1956–1961), Austin, TX, USA.
[103] Henni, D. E., Hadjaj-Aoul, Y., and Ghomari, A. (2017). Probe-SDN: A smart monitoring framework for SDN-based networks. In the proceedings of the Global Information Infrastructure and Networking Symposium (pp. 1–6), France.
[104] Hauser, F., Haberle, M., Merling, D., Lindner, S., Gurevich, V., Zeiger, F., Frank, R. and Menth, M. (2021, January). A Survey on Data Plane Programming with P4: Fundamentals, Advances and Applied Research. arXiv:2101.10632v1 [cs.NI].
[105] Moyano, R. F., Fernandez, D., Merayo, N., Lentisco, C. M. and Cardenas, A. (2020, Feb.). NFV and SDN-Based Differentiated Traffic Treatment for Residential Networks. IEEE Access.
[106] Sherwood, R., Chan, M., Covington, A., Gibb, G., Flajslik, M., Handigol, N., Huang, T.-Y., Kazemian, P., Kobayashi, M., Naous, J., Seetharaman, S., Underhill, D., Yabe, T., Yap, K.-K., Yiakoumis, Y., Zeng, H., Appenzeller, G., Johari, R., McKeown, N. and Parulkar, G. (2010, Jan.) .Carving research slices out of your production networks with openflow. SIGCOMM Computer Communication Review (pp. 129-130), 40(1).
[107] Al-Shabibi, A., Leenheer, M. D., Gerola, M., Koshibe, A., Snow, W., and Parulkar, G. (2014). Openvirtex: A network hypervisor. In Open Networking Summit SantaClara, CA: USENIX Association.
[108] Drutskoy, D., Keller, E., and Rexford, J., (2013, March). Scalable network virtualization in software-defined networks. IEEE Internet Computing (pp. 20-27), 17(2).
[109] Koponen, T., Amidon, K., Balland, P., Casado, M., Chanda, A., Fulton, B., Ganichev, I., Gross, J., Ingram, P., Jackson, E., A. Lambeth, Lenglet, R., Li, S. H., Padmanabhan, A., Pettit, J., Ramanathan, R., Shenker, S., Shieh, A., Stribling, J., Thakkar, P., Wendlandt, D., Yip, A., and Zhang, R. (2014). Network virtualization in multi-tenant datacenters. In Eleventh USENIX Symposium on Networked Systems Design and Implementation (NSDI 14) (pp. 203-216). Seattle, USENIX Association.
[110] Turull, D., Hidell, M., and odin, P. S. (2012, June). libnetvirt: The network virtualization library. In IEEE International Conference on Communications (ICC) (pp. 5543-5547).
[111] Basta, A., Kellerer, W., Hoffmann, M., Hoffmann, K. and Schmidt, E. E. (2013, Nov.). A virtual SDN-enabled LTE EPC Archtiecture: A case study for S/P-Gateways Functions. In IEEE future Networks and Services (pp.1-7).
[112] Osiński, T., Tarasiuk, H., Chaignon, P., Kossakowski, M. (2020, June). P4rt-OVS: Programming Protocol-Independent, Runtime Extensions for Open vSwitch with P4. IEEE IFIP Networking Conference (Networking) (pp. 22-26).
[113] Hsu, K. F., Beckett, R., Chen, A., Rexford, J., Tamanna, P., Walker, B. (2020). Contra: A Programmable System for Performance-Aware Routing. In the Proceedings of Seventeenth USENIX Symposium on Networked Systems Design and Implementation, CA, USA.
[114] Kloti, R., Kotronis, V., and Smith, P. (2013). OpenFlow: A security analysis. In Proceedings of Twenty first IEEE International Conference on Network Protocols (ICNP) (pp. 1-6).
[115] Uncover Security Design Flaws Using STRIDE Approach, [Online]. Available: https://adam.shostack.org/uncover.html
[116] OpenDaylight AAA Project. [Online] Available: https://wiki.opendaylight.org/view/AAA:Main
[117] Oktian, Y. E., Lee, S., Lee, H. and Lam, J. (2015, July). Secure your northbound SDN API. In proceedings of Seventh International Conference on Ubiquitous and Future Networks (pp. 919-920).
[118] Hardt, D. The OAuth 2.0 Authorization Framework. [Online] Available: https://tools.ietf.org/html/rfc6749

Authors' Profiles

Suhail Ahmad Mir received B.E degree in Computer Science & Engineering from the University of Kashmir, India, in 2008 and the M. Tech degree in Communication and Information Technology from National Institute of Technology, Srinagar, Jammu and Kashmir, in 2011. He has been working as Assistant Professor in Department of Computer Science and Engineering, University of Kashmir since 2012. Currently pursuing Ph.D. at NIT Srinagar and his research interests include Network Security, MPLS and Software Defined Networks.

Ajaz Hussain Mir is Professor and Head of Electronics and Communication Department, National Institute of Technology, Srinagar, Jammu and Kashmir. He received the Ph. D and M. Tech degree from IIT-Delhi. His current research interests include Digital Image Processing, Network Security, Mobile Networks, IOT, Next Generation Networks and Software Defined Networks.

How to cite this paper: Suhail Ahmad, Ajaz Hussain Mir, " SDN Interfaces: Protocols, Taxonomy and Challenges", International Journal of Wireless and Microwave Technologies(IJWMT), Vol.12, No.2, pp. 11-32, 2022.DOI: 10.5815/ijwmt.2022.02.02