Internet Research Task Force (IRTF)
Request for Comments: 7426
Category: Informational
ISSN: 2070-1721

E. Haleplidis, Ed.
University of Patras
K. Pentikousis, Ed.
EICT
S. Denazis
University of Patras
J. Hadi Salim
Mojatatu Networks
D. Meyer
Brocade
O. Koufopavlou
University of Patras

January 2015

Software-Defined Networking (SDN): Layers and Architecture Terminology

Abstract

Software-Defined Networking (SDN) refers to a new approach for network programmability, that is, the capacity to initialize, control, change, and manage network behavior dynamically via open interfaces. SDN emphasizes the role of software in running networks through the introduction of an abstraction for the data forwarding plane and, by doing so, separates it from the control plane. This separation allows faster innovation cycles at both planes as experience has already shown. However, there is increasing confusion as to what exactly SDN is, what the layer structure is in an SDN architecture, and how layers interface with each other. This document, a product of the IRTF Software-Defined Networking Research Group (SDNRG), addresses these questions and provides a concise reference for the SDN research community based on relevant peer-reviewed literature, the RFC series, and relevant documents by other standards organizations.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Research Task Force (IRTF). The IRTF publishes the results of Internet-related research and development activities. These results might not be suitable for deployment. This RFC represents the consensus of the Software-Defined Networking Research Group of the Internet Research Task Force (IRTF). Documents approved for publication by the IRSG are not a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7426.

Copyright Notice

Table of Contents

1. Introduction

"Software-Defined Networking (SDN)" is a term of the programmable networks paradigm [PNSurvey99] [OF08]. In short, SDN refers to the ability of software applications to program individual network devices dynamically and therefore control the behavior of the network as a whole [NV09]. Boucadair and Jacquenet [RFC7149] point out that SDN is a set of techniques used to facilitate the design, delivery, and operation of network services in a deterministic, dynamic, and scalable manner.

A key element in SDN is the introduction of an abstraction between the (traditional) forwarding and control planes in order to separate them and provide applications with the means necessary to programmatically control the network. The goal is to leverage this separation, and the associated programmability, in order to reduce complexity and enable faster innovation at both planes [A4D05].

The historical evolution of the research and development area of programmable networks is reviewed in detail in [SDNHistory] [SDNSurvey], starting with efforts dating back to the 1980s. As documented in [SDNHistory], many of the ideas, concepts, and concerns are applicable to the latest research and development in SDN (and SDN standardization) and have been under extensive investigation and discussion in the research community for quite some time. For example, Rooney, et al. [Tempest] discuss how to allow third-party access to the network without jeopardizing network integrity or how to accommodate legacy networking solutions in their (then new) programmable environment. Further, the concept of separating the control and forwarding planes, which is prominent in SDN, has been extensively discussed even prior to 1998 [Tempest] [P1520] in SS7 networks [ITUSS7], Ipsilon Flow Switching [RFC1953] [RFC2297], and ATM [ITUATM].

SDN research often focuses on varying aspects of programmability, and we are frequently confronted with conflicting points of view regarding what exactly SDN is. For instance, we find that for various reasons (e.g., work focusing on one domain and therefore not necessarily applicable as-is to other domains), certain well-accepted definitions do not correlate well with each other. For example, both OpenFlow [OpenFlow] and the Network Configuration Protocol (NETCONF) [RFC6241] have been characterized as SDN interfaces, but they refer to control and management, respectively.

This motivates us to consolidate the definitions of SDN in the literature and correlate them with earlier work at the IETF and the research community. Of particular interest is, for example, to determine which layers comprise the SDN architecture and which

interfaces and their corresponding attributes are best suited to be used between them. As such, the aim of this document is not to standardize any particular layer or interface but rather to provide a concise reference that reflects current approaches regarding the SDN layer architecture. We expect that this document would be useful to upcoming work in SDNRG as well as future discussions within the SDN community as a whole.

This document addresses the work item in the SDNRG charter titled "Survey of SDN approaches and Taxonomies", fostering better understanding of prominent SDN technologies in a technology-impartial and business-agnostic manner but does not constitute a new IETF standard. It is meant as a common base for further discussion. As such, we do not make any value statements nor discuss the applicability of any of the frameworks examined in this document for any particular purpose. Instead, we document their characteristics and attributes and classify them, thus providing a taxonomy. This document does not intend to provide an exhaustive list of SDN research issues; interested readers should consider reviewing [SLTSDN] and [SDNACS]. In particular, Jarraya, et al. [SLTSDN] provide an overview of SDN-related research topics, e.g., control partitioning, which is related to the Consistency, Availability and Partitioning (CAP) theorem discussed in Section 3.5.4.

This document has been extensively reviewed, discussed, and commented by the vast majority of SDNRG members, a community that certainly exceeds 100 individuals. It is the consensus of SDNRG that this document should be published in the IRTF stream of the RFC series [RFC5743].

The remainder of this document is organized as follows. Section 2 explains the terminology used in this document. Section 3 introduces a high-level overview of current SDN architecture abstractions. Finally, Section 4 discusses how the SDN layer architecture relates to prominent SDN-enabling technologies.

2. Terminology

This document uses the following terms:

- Software-Defined Networking (SDN) - A programmable networks approach that supports the separation of control and forwarding planes via standardized interfaces.

- Resource - A physical or virtual component available within a system. Resources can be very simple or fine-grained (e.g., a port or a queue) or complex, comprised of multiple resources (e.g., a network device).

- Network Device - A device that performs one or more network operations related to packet manipulation and forwarding. This reference model makes no distinction whether a network device is physical or virtual. A device can also be considered as a container for resources and can be a resource in itself.

- Interface - A point of interaction between two entities. When the entities are placed at different locations, the interface is usually implemented through a network protocol. If the entities are collocated in the same physical location, the interface can be implemented using a software application programming interface (API), inter-process communication (IPC), or a network protocol.

- Application (App) - An application in the context of SDN is a piece of software that utilizes underlying services to perform a function. Application operation can be parameterized, for example, by passing certain arguments at call time, but it is meant to be a standalone piece of software; an App does not offer any interfaces to other applications or services.

- Service - A piece of software that performs one or more functions and provides one or more APIs to applications or other services of the same or different layers to make use of said functions and returns one or more results. Services can be combined with other services, or called in a certain serialized manner, to create a new service.

- Forwarding Plane (FP) - The collection of resources across all network devices responsible for forwarding traffic.

- Operational Plane (OP) - The collection of resources responsible for managing the overall operation of individual network devices.

- Control Plane (CP) - The collection of functions responsible for controlling one or more network devices. CP instructs network devices with respect to how to process and forward packets. The control plane interacts primarily with the forwarding plane and, to a lesser extent, with the operational plane.

- Management Plane (MP) - The collection of functions responsible for monitoring, configuring, and maintaining one or more network devices or parts of network devices. The management plane is mostly related to the operational plane (it is related less to the forwarding plane).

- Application Plane - The collection of applications and services that program network behavior.

- Device and resource Abstraction Layer (DAL) - The device's resource abstraction layer based on one or more models. If it is a physical device, it may be referred to as the Hardware Abstraction Layer (HAL). DAL provides a uniform point of reference for the device's forwarding- and operational-plane resources.

- Control Abstraction Layer (CAL) - The control plane's abstraction layer. CAL provides access to the Control-Plane Southbound Interface.

- Management Abstraction Layer (MAL) - The management plane's abstraction layer. MAL provides access to the Management-Plane Southbound Interface.

- Network Services Abstraction Layer (NSAL) - Provides service abstractions that can be used by applications and services.

3. SDN Layers and Architecture

Figure 1 summarizes the SDN architecture abstractions in the form of a detailed, high-level schematic. Note that in a particular implementation, planes can be collocated with other planes or can be physically separated, as we discuss below.

SDN is based on the concept of separation between a controlled entity and a controller entity. The controller manipulates the controlled entity via an interface. Interfaces, when local, are mostly API invocations through some library or system call. However, such interfaces may be extended via some protocol definition, which may use local inter-process communication (IPC) or a protocol that could also act remotely; the protocol may be defined as an open standard or in a proprietary manner.

Day [PiNA] explores the use of IPC as the mainstay for the definition of recursive network architectures with varying degrees of scope and range of operation. The Recursive InterNetwork Architecture [RINA] outlines a recursive network architecture based on IPC that capitalizes on repeating patterns and structures. This document does not propose a new architecture -- we simply document previous work through a taxonomy. Although recursion is out of the scope of this work, Figure 1 illustrates a hierarchical model in which layers can be stacked on top of each other and employed recursively as needed.

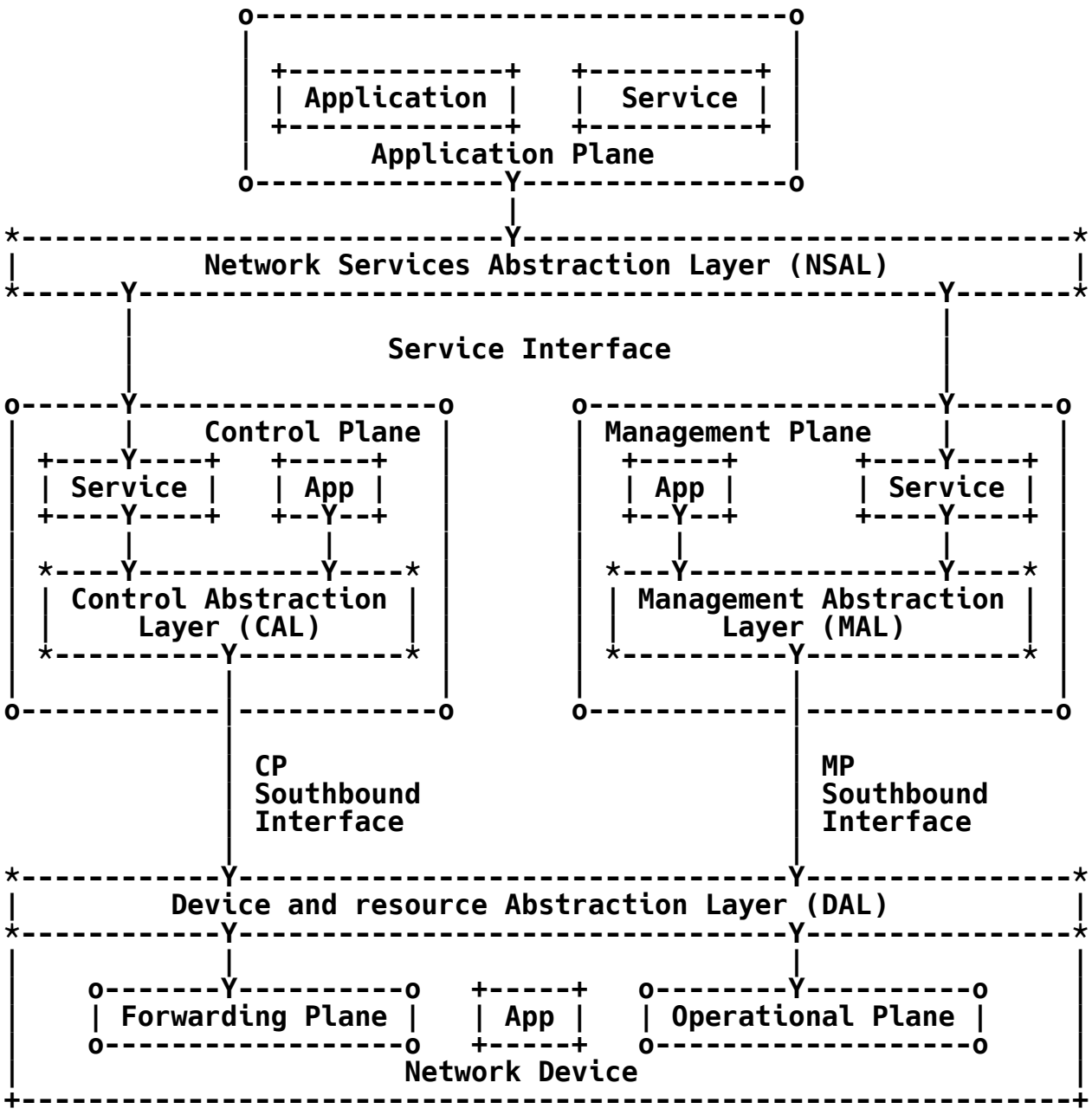

Figure 1: SDN Layer Architecture

3.1. Overview

This document follows a network-device-centric approach: control mostly refers to the device packet-handling capability, while management typically refers to aspects of the overall device operation. We view a network device as a complex resource that contains and is part of multiple resources similar to [DIOPR]. Resources can be simple, single components of a network device, for example, a port or a queue of the device, and can also be aggregated into complex resources, for example, a network card or a complete network device.

The reader should keep in mind that we make no distinction between "physical" and "virtual" resources or "hardware" and "software" realizations in this document, as we do not delve into implementation or performance aspects. In other words, a resource can be implemented fully in hardware, fully in software, or any hybrid combination in between. Further, we do not distinguish whether a resource is implemented as an overlay or as a part/component of some other device. In general, network device software can run on so-called "bare metal" or on a virtualized substrate. Finally, this document does not discuss how resources are allocated, orchestrated, and released. Indeed, orchestration is out of the scope of this document.

SDN spans multiple planes as illustrated in Figure 1. Starting from the bottom part of the figure and moving towards the upper part, we identify the following planes:

- Forwarding Plane - Responsible for handling packets in the data path based on the instructions received from the control plane. Actions of the forwarding plane include, but are not limited to, forwarding, dropping, and changing packets. The forwarding plane is usually the termination point for control-plane services and applications. The forwarding plane can contain forwarding resources such as classifiers. The forwarding plane is also widely referred to as the "data plane" or the "data path".

- Operational Plane - Responsible for managing the operational state of the network device, e.g., whether the device is active or inactive, the number of ports available, the status of each port, and so on. The operational plane is usually the termination point for management-plane services and applications. The operational plane relates to network device resources such as ports, memory, and so on. We note that some participants of the IRTF SDNRG have a different opinion in regards to the definition of the operational plane. That is, one can argue that the operational plane does not constitute a "plane" per se, but it is, in

practice, an amalgamation of functions on the forwarding plane. For others, however, a "plane" allows one to distinguish between different areas of operations; therefore, the operational plane is included as a "plane" in Figure 1. We have adopted this latter view in this document.

- Control Plane - Responsible for making decisions on how packets should be forwarded by one or more network devices and pushing such decisions down to the network devices for execution. The control plane usually focuses mostly on the forwarding plane and less on the operational plane of the device. The control plane may be interested in operational-plane information, which could include, for instance, the current state of a particular port or its capabilities. The control plane's main job is to fine-tune the forwarding tables that reside in the forwarding plane, based on the network topology or external service requests.

- Management Plane - Responsible for monitoring, configuring, and maintaining network devices, e.g., making decisions regarding the state of a network device. The management plane usually focuses mostly on the operational plane of the device and less on the forwarding plane. The management plane may be used to configure the forwarding plane, but it does so infrequently and through a more wholesale approach than the control plane. For instance, the management plane may set up all or part of the forwarding rules at once, although such action would be expected to be taken sparingly.

- Application Plane - The plane where applications and services that define network behavior reside. Applications that directly (or primarily) support the operation of the forwarding plane (such as routing processes within the control plane) are not considered part of the application plane. Note that applications may be implemented in a modular and distributed fashion and, therefore, can often span multiple planes in Figure 1.

[RFC7276] has defined the data, control, and management planes in terms of Operations, Administration, and Maintenance (OAM). This document attempts to broaden the terms defined in [RFC7276] in order to reflect all aspects of an SDN architecture.

All planes mentioned above are connected via interfaces (indicated with "Y" in Figure 1. An interface may take multiple roles depending on whether the connected planes reside on the same (physical or virtual) device. If the respective planes are designed so that they do not have to reside in the same device, then the interface can only take the form of a protocol. If the planes are collocated on the

same device, then the interface could be implemented via an open/proprietary protocol, an open/proprietary software inter-process communication API, or operating system kernel system calls.

Applications, i.e., software programs that perform specific computations that consume services without providing access to other applications, can be implemented natively inside a plane or can span multiple planes. For instance, applications or services can span both the control and management planes and thus be able to use both the Control-Plane Southbound Interface (CPSI) and Management-Plane Southbound Interface (MPSI), although this is only implicitly illustrated in Figure 1. An example of such a case would be an application that uses both [OpenFlow] and [OF-CONFIG].

Services, i.e., software programs that provide APIs to other applications or services, can also be natively implemented in specific planes. Services that span multiple planes belong to the application plane as well.

While not shown explicitly in Figure 1, services, applications, and entire planes can be placed in a recursive manner, thus providing overlay semantics to the model. For example, application-plane services can be provided to other applications or services through NSAL. Additional examples include virtual resources that are realized on top of a physical resources and hierarchical control-plane controllers [KANDOO].

Note that the focus in this document is, of course, on the north/south communication between entities in different planes. But this, clearly, does not exclude entity communication within any one plane.

It must be noted, however, that in Figure 1, we present an abstract view of the various planes, which is devoid of implementation details. Many implementations in the past have opted for placing the management plane on top of the control plane. This can be interpreted as having the control plane acting as a service to the management plane. Further, in many networks, especially in Internet routers and Ethernet switches, the control plane has been usually implemented as tightly coupled with the network device. When taken as a whole, the control plane has been distributed network-wide. On the other hand, the management plane has been traditionally centralized and has been responsible for managing the control plane and the devices. However, with the adoption of SDN principles, this distinction is no longer so clear-cut.

Additionally, this document considers four abstraction layers:

- The Device and resource Abstraction Layer (DAL) abstracts the resources of the device's forwarding and operational planes to the control and management planes. Variations of DAL may abstract both planes or either of the two and may abstract any plane of the device to either the control or management plane.

- The Control Abstraction Layer (CAL) abstracts the Control-Plane Southbound Interface and the DAL from the applications and services of the control plane.

- The Management Abstraction Layer (MAL) abstracts the Management-Plane Southbound Interface and the DAL from the applications and services of the management plane.

- The Network Services Abstraction Layer (NSAL) provides service abstractions for use by applications and other services.

At the time of this writing, SDN-related activities have begun in other SDOs. For example, at the ITU, work on architectural [ITUSG13] and signaling requirements and protocols [ITUSG11] has commenced, but the respective study groups have yet to publish their documents, with the exception of [ITUY3300]. The views presented in [ITUY3300] as well as in [ONFArch] are well aligned with this document.

3.2. Network Devices

A network device is an entity that receives packets on its ports and performs one or more network functions on them. For example, the network device could forward a received packet, drop it, alter the packet header (or payload), forward the packet, and so on. A network device is an aggregation of multiple resources such as ports, CPU, memory, and queues. Resources are either simple or can be aggregated to form complex resources that can be viewed as one resource. The network device is in itself a complex resource. Examples of network devices include switches and routers. Additional examples include network elements that may operate at a layer above IP (such as firewalls, load balancers, and video transcoders) or below IP (such as Layer 2 switches and optical or microwave network elements).

Network devices can be implemented in hardware or software and can be either physical or virtual. As has already been mentioned before, this document makes no such distinction. Each network device has a presence in a forwarding plane and an operational plane.

The forwarding plane, commonly referred to as the "data path", is responsible for handling and forwarding packets. The forwarding plane provides switching, routing, packet transformation, and filtering functions. Resources of the forwarding plane include but are not limited to filters, meters, markers, and classifiers.

The operational plane is responsible for the operational state of the network device, for instance, with respect to status of network ports and interfaces. Operational-plane resources include, but are not limited to, memory, CPU, ports, interfaces, and queues.

The forwarding and the operational planes are exposed via the Device and resource Abstraction Layer (DAL), which may be expressed by one or more abstraction models. Examples of forwarding-plane abstraction models are Forwarding and Control Element Separation (ForCES) [RFC5812], OpenFlow [OpenFlow], YANG model [RFC6020], and SNMP MIBs [RFC3418]. Examples of the operational-plane abstraction model include the ForCES model [RFC5812], the YANG model [RFC6020], and SNMP MIBs [RFC3418].

Note that applications can also reside in a network device. Examples of such applications include event monitoring and handling (offloading) topology discovery or ARP [RFC0826] in the device itself instead of forwarding such traffic to the control plane.

3.3. Control Plane

The control plane is usually distributed and is responsible mainly for the configuration of the forwarding plane using a Control-Plane Southbound Interface (CPSI) with DAL as a point of reference. CP is responsible for instructing FP about how to handle network packets.

Communication between control-plane entities, colloquially referred to as the "east-west" interface, is usually implemented through gateway protocols such as BGP [RFC4271] or other protocols such as the Path Computation Element (PCE) Communication Protocol (PCEP) [RFC5440]. These corresponding protocol messages are usually exchanged in-band and subsequently redirected by the forwarding plane to the control plane for further processing. Examples in this category include [RCP], [SoftRouter], and [RouteFlow].

Control-plane functionalities usually include:

- Topology discovery and maintenance
- Packet route selection and instantiation
- Path failover mechanisms

The CPSI is usually defined with the following characteristics:

- time-critical interface that requires low latency and sometimes high bandwidth in order to perform many operations in short order
- oriented towards wire efficiency and device representation instead of human readability

Examples include fast- and high-frequency of flow or table updates, high throughput, and robustness for packet handling and events.

CPSI can be implemented using a protocol, an API, or even inter-process communication. If the control plane and the network device are not collocated, then this interface is certainly a protocol. Examples of CPSIs are ForCES [RFC5810] and the OpenFlow protocol [OpenFlow].

The Control Abstraction Layer (CAL) provides access to control applications and services to various CPSIs. The control plane may support more than one CPSI.

Control applications can use CAL to control a network device without providing any service to upper layers. Examples include applications that perform control functions, such as OSPF, IS-IS, and BGP.

Control-plane service examples include a virtual private LAN service, service tunnels, topology services, etc.

3.4. Management Plane

The management plane is usually centralized and aims to ensure that the network as a whole is running optimally by communicating with the network devices' operational plane using a Management-Plane Southbound Interface (MPSI) with DAL as a point of reference.

Management-plane functionalities are typically initiated, based on an overall network view, and traditionally have been human-centric. However, lately, algorithms are replacing most human intervention. Management-plane functionalities [FCAPS] typically include:

- Fault and monitoring management
- Configuration management

In addition, management-plane functionalities may also include entities such as orchestrators, Virtual Network Function Managers (VNF Managers) and Virtualised Infrastructure Managers, as described in [NFVArch]. Such entities can use management interfaces to

operational-plane resources to request and provision resources for virtual functions as well as instruct the instantiation of virtual forwarding functions on top of physical forwarding functions. The possibility of a common abstraction model for both SDN and Network Function Virtualization (NFV) is explored in [SDNNFV]. Note, however, that these are only examples of applications and services in the management plane and not formal definitions of entities in this document. As has been noted above, orchestration and therefore the definition of any associated entities is out of the scope of this document.

The MPSI, in contrast to the CPSI, is usually not a time-critical interface and does not share the CPSI requirements.

MPSI is typically closer to human interaction than CPSI (cf. [RFC3535]); therefore, MPSI usually has the following characteristics:

- It is oriented more towards usability, with optimal wire performance being a secondary concern.
- Messages tend to be less frequent than in the CPSI.

As an example of usability versus performance, we refer to the consensus of the 2002 IAB Workshop [RFC3535]: the key requirement for a network management technology is ease of use, not performance. As per [RFC6632], textual configuration files should be able to contain international characters. Human-readable strings should utilize UTF-8, and protocol elements should be in case-insensitive ASCII, which requires more processing capabilities to parse.

MPSI can range from a protocol, to an API or even inter-process communication. If the management plane is not embedded in the network device, the MPSI is certainly a protocol. Examples of MPSIs are ForCES [RFC5810], NETCONF [RFC6241], IP Flow Information Export (IPFIX) [RFC7011], Syslog [RFC5424], Open vSwitch Database (OVSDB) [RFC7047], and SNMP [RFC3411].

The Management Abstraction Layer (MAL) provides access to management applications and services to various MPSIs. The management plane may support more than one MPSI.

Management applications can use MAL to manage the network device without providing any service to upper layers. Examples of management applications include network monitoring, fault detection, and recovery applications.

Management-plane services provide access to other services or applications above the management plane.

3.5. Discussion of Control and Management Planes

The definition of a clear distinction between "control" and "management" in the context of SDN received significant community attention during the preparation of this document. We observed that the role of the management plane has been earlier largely ignored or specified as out-of-scope for the SDN ecosystem. In the remainder of this subsection, we summarize the characteristics that differentiate the two planes in order to have a clear understanding of the mechanics, capabilities, and needs of each respective interface.

3.5.1. Timescale

A point has been raised regarding the reference timescales for the control and management planes regarding how fast the respective plane is required to react to, or how fast it needs to manipulate, the forwarding or operational plane of the device. In general, the control plane needs to send updates "often", which translates roughly to a range of milliseconds; that requires high-bandwidth and low-latency links. In contrast, the management plane reacts generally at longer time frames, i.e., minutes, hours, or even days; thus, wire efficiency is not always a critical concern. A good example of this is the case of changing the configuration state of the device.

3.5.2. Persistence

Another distinction between the control and management planes relates to state persistence. A state is considered ephemeral if it has a very limited lifespan and is not deemed necessary to be stored on non-volatile memory. A good example is determining routing, which is usually associated with the control plane. On the other hand, a persistent state has an extended lifespan that may range from hours to days and months, is meant to be used beyond the lifetime of the process that created it, and is thus used across device reboots. Persistent state is usually associated with the management plane.

3.5.3. Locality

As mentioned earlier, traditionally, the control plane has been executed locally on the network device and is distributed in nature whilst the management plane is usually executed in a centralized manner, remotely from the device. However, with the advent of SDN centralizing, or "logically centralizing", the controller tends to muddle the distinction of the control and management plane based on locality.

3.5.4. CAP Theorem Insights

The CAP theorem views a distributed computing system as composed of multiple computational resources (i.e., CPU, memory, storage) that are connected via a communications network and together perform a task. The theorem, or conjecture by some, identifies three characteristics of distributed systems that are universally desirable:

- Consistency, meaning that the system responds identically to a query no matter which node receives the request (or does not respond at all).
- Availability, i.e., that the system always responds to a request (although the response may not be consistent or correct).
- Partition tolerance, namely that the system continues to function even when specific nodes or the communications network fail.

In 2000, Eric Brewer [CAPBR] conjectured that a distributed system can satisfy any two of these guarantees at the same time but not all three. This conjecture was later proven by Gilbert and Lynch [CAPGL] and is now usually referred to as the CAP theorem [CAPFN].

Forwarding a packet through a network correctly is a computational problem. One of the major abstractions that SDN posits is that all network elements are computational resources that perform the simple computational task of inspecting fields in an incoming packet and deciding how to forward it. Since the task of forwarding a packet from network ingress to network egress is obviously carried out by a large number of forwarding elements, the network of forwarding devices is a distributed computational system. Hence, the CAP theorem applies to forwarding of packets.

In the context of the CAP theorem, if one considers partition tolerance of paramount importance, traditional control-plane operations are usually local and fast (available), while management-plane operations are usually centralized (consistent) and may be slow.

The CAP theorem also provides insights into SDN architectures. For example, a centralized SDN controller acts as a consistent global database and specific SDN mechanisms ensure that a packet entering the network is handled consistently by all SDN switches. The issue of tolerance to loss of connectivity to the controller is not addressed by the basic SDN model. When an SDN switch cannot reach its controller, the flow will be unavailable until the connection is restored. The use of multiple non-collocated SDN controllers has

been proposed (e.g., by configuring the SDN switch with a list of controllers); this may improve partition tolerance but at the cost of loss of absolute consistency. Panda, et al. [CAPFN] provide a first exploration of how the CAP theorem applies to SDN.

3.6. Network Services Abstraction Layer

The Network Services Abstraction Layer (NSAL) provides access from services of the control, management, and application planes to other services and applications. We note that the term "SAL" is overloaded, as it is often used in several contexts ranging from system design to service-oriented architectures; therefore, we explicitly add "Network" to the title of this layer to emphasize that this term relates to Figure 1, and we map it accordingly in Section 4 to prominent SDN approaches.

Service interfaces can take many forms pertaining to their specific requirements. Examples of service interfaces include, but are not limited to, RESTful APIs, open protocols such as NETCONF, inter-process communication, CORBA [CORBA] interfaces, and so on. The two leading approaches for service interfaces are RESTful interfaces and Remote Procedure Call (RPC) interfaces. Both follow a client-server architecture and use XML or JSON to pass messages, but each has some slightly different characteristics.

RESTful interfaces, designed according to the representational state transfer design paradigm [REST], have the following characteristics:

- Resource identification - Individual resources are identified using a resource identifier, for example, a URI.
- Manipulation of resources through representations - Resources are represented in a format like JSON, XML, or HTML.
- Self-descriptive messages - Each message has enough information to describe how the message is to be processed.
- Hypermedia as the engine of application state - A client needs no prior knowledge of how to interact with a server, as the API is not fixed but dynamically provided by the server.

Remote procedure calls (RPCs) [RFC5531], e.g., XML-RPC and the like, have the following characteristics:

- Individual procedures are identified using an identifier.
- A client needs to know the procedure name and the associated parameters.

3.7. Application Plane

Applications and services that use services from the control and/or management plane form the application plane.

Additionally, services residing in the application plane may provide services to other services and applications that reside in the application plane via the service interface.

Examples of applications include network topology discovery, network provisioning, path reservation, etc.

4. SDN Model View

We advocate that the SDN southbound interface should encompass both CPSI and MPSI.

SDN controllers such as [NOX] and [Beacon] are a collection of control-plane applications and services that implement a CPSI ([NOX] and [Beacon] both use OpenFlow) and provide a northbound interface for applications. The SDN northbound interface for controllers is implemented in the Network Services Abstraction Layer (NSAL) of Figure 1.

The above model can be used to describe all prominent SDN-enabling technologies in a concise manner, as we explain in the following subsections.

4.1. ForCES

The IETF Forwarding and Control Element Separation (ForCES) framework [RFC3746] consists of one model and two protocols. ForCES separates the forwarding plane from the control plane via an open interface, namely the ForCES protocol [RFC5810], which operates on entities of the forwarding plane that have been modeled using the ForCES model [RFC5812].

The ForCES model [RFC5812] is based on the fact that a network element is composed of numerous logically separate entities that cooperate to provide a given functionality (such as routing or IP switching) and yet appear as a normal integrated network element to external entities.

ForCES models the forwarding plane using Logical Functional Blocks (LFBs), which, when connected in a graph, compose the Forwarding Element (FE). LFBs are described in XML, based on an XML schema.

LFB definitions include base and custom-defined datatypes; metadata definitions; input and output ports; operational parameters or components; and capabilities and event definitions.

The ForCES model can be used to define LFBs from fine- to coarse-grained as needed, irrespective of whether they are physical or virtual.

The ForCES protocol is agnostic to the model and can be used to monitor, configure, and control any ForCES-modeled element. The protocol has very simple commands: Set, Get, and Del(ete). The ForCES protocol has been designed for high throughput and fast updates.

With respect to Figure 1, the ForCES model [RFC5812] is suitable for the DAL, both for the operational and the forwarding plane, using LFBs. The ForCES protocol [RFC5810] has been designed and is suitable for the CPSI, although it could also be utilized for the MPSI.

4.2. NETCONF/YANG

The Network Configuration Protocol (NETCONF) [RFC6241] is an IETF network management protocol [RFC6632]. NETCONF provides mechanisms to install, manipulate, and delete the configuration of network devices.

NETCONF protocol operations are realized as remote procedure calls (RPCs). The NETCONF protocol uses XML-based data encoding for the configuration data as well as the protocol messages. Recent studies, such as [ESNet] and [PENet], have shown that NETCONF performs better than SNMP [RFC3411].

Additionally, the YANG data modeling language [RFC6020] has been developed for specifying NETCONF data models and protocol operations. YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol, NETCONF remote procedure calls, and NETCONF notifications.

YANG models the hierarchical organization of data as a tree, in which each node has either a value or a set of child nodes. Additionally, YANG structures data models into modules and submodules, allowing reusability and augmentation. YANG models can describe constraints to be enforced on the data. Additionally, YANG has a set of base datatypes and allows custom-defined datatypes as well.

YANG allows the definition of NETCONF RPCs, which allows the protocol to have an extensible number of commands. For RPC definitions, the operations names, input parameters, and output parameters are defined using YANG data definition statements.

With respect to Figure 1, the YANG model [RFC6020] is suitable for specifying DAL for the forwarding and operational planes. NETCONF [RFC6241] is suitable for the MPSI. NETCONF is a management protocol [RFC6632], which was not (originally) designed for fast CP updates, and it might not be suitable for addressing the requirements of CPSI.

4.3. OpenFlow

OpenFlow is a framework originally developed at Stanford University and currently under active standards development [OpenFlow] through the Open Networking Foundation (ONF). Initially, the goal was to provide a way for researchers to run experimental protocols in a production network [OF08]. OpenFlow has undergone many revisions, and additional revisions are likely. The following description reflects version 1.4 [OpenFlow]. In short, OpenFlow defines a protocol through which a logically centralized controller can control an OpenFlow switch. Each OpenFlow-compliant switch maintains one or more flow tables, which are used to perform packet lookups. Distinct actions are to be taken regarding packet lookup and forwarding. A group table and an OpenFlow channel to external controllers are also part of the switch specification.

With respect to Figure 1, the OpenFlow switch specifications [OpenFlow] define a DAL for the forwarding plane as well as for CPSI. The OF-CONFIG protocol [OF-CONFIG], based on the YANG model [RFC6020], provides a DAL for the forwarding and operational planes of an OpenFlow switch and specifies NETCONF [RFC6241] as the MPSI. OF-CONFIG overlaps with the OpenFlow DAL, but with NETCONF [RFC6241] as the transport protocol, it shares the limitations described in the previous section.

4.4. Interface to the Routing System

Interface to the Routing System (I2RS) provides a standard interface to the routing system for real-time or event-driven interaction through a collection of protocol-based control or management interfaces. Essentially, one of the main goals of I2RS, is to make the Routing Information Base (RIB) programmable, thus enabling new kinds of network provisioning and operation.

I2RS did not initially intend to create new interfaces but rather leverage or extend existing ones and define informational models for the routing system. For example, the latest I2RS problem statement

[I2RSProb] discusses previously defined IETF protocols such as ForCES [RFC5810], NETCONF [RFC6241], and SNMP [RFC3417]. Regarding the definition of informational and data models, the I2RS working group has opted to use the YANG [RFC6020] modeling language.

Currently the I2RS working group is developing an Information Model [I2RSInfo] in regards to the Network Services Abstraction Layer for the I2RS agent.

With respect to Figure 1, the I2RS architecture [I2RSArch] encompasses the control and application planes and uses any CPSI and DAL that is available, whether that may be ForCES [RFC5810], OpenFlow [OpenFlow], or another interface. In addition, the I2RS agent is a control-plane service. All services or applications on top of that belong to either the Control, Management, or Application plane. In the I2RS documents, management access to the agent may be provided by management protocols like SNMP and NETCONF. The I2RS protocol may also be mapped to the service interface as it will even provide access to services and applications other than control-plane services and applications.

4.5. SNMP

The Simple Network Management Protocol (SNMP) is an IETF-standardized management protocol and is currently at its third revision (SNMPv3) [RFC3417] [RFC3412] [RFC3414]. It consists of a set of standards for network management, including an application-layer protocol, a database schema, and a set of data objects. SNMP exposes management data (managed objects) in the form of variables on the managed systems, which describe the system configuration. These variables can then be queried and set by managing applications.

SNMP uses an extensible design for describing data, defined by Management Information Bases (MIBs). MIBs describe the structure of the management data of a device subsystem. MIBs use a hierarchical namespace containing object identifiers (OIDs). Each OID identifies a variable that can be read or set via SNMP. MIBs use the notation defined by Structure of Management Information Version 2 [RFC2578].

An early example of SNMP in the context of SDN is discussed in [Peregrine].

With respect to Figure 1, SNMP MIBs can be used to describe DAL for the forwarding and operational planes. Similar to YANG, SNMP MIBs are able to describe DAL for the forwarding plane. SNMP, similar to NETCONF, is suited for the MPSI.

4.6. PCEP

The Path Computation Element (PCE) [RFC4655] architecture defines an entity capable of computing paths for a single service or a set of services. A PCE might be a network node, network management station, or dedicated computational platform that is resource-aware and has the ability to consider multiple constraints for a variety of path computation problems and switching technologies. The PCE Communication Protocol (PCEP) [RFC5440] is used between a Path Computation Client (PCC) and a PCE, or between multiple PCEs.

The PCE architecture represents a vision of networks that separates path computation for services, the signaling of end-to-end connections, and actual packet forwarding. The definition of online and offline path computation is dependent on the reachability of the PCE from network and Network Management System (NMS) nodes and the type of optimization request that may significantly impact the optimization response time from the PCE to the PCC.

The PCEP messaging mechanism facilitates the specification of computation endpoints (source and destination node addresses), objective functions (requested algorithm and optimization criteria), and the associated constraints such as traffic parameters (e.g., requested bandwidth), the switching capability, and encoding type.

With respect to Figure 1, PCE is a control-plane service that provides services for control-plane applications. PCEP may be used as an east-west interface between PCEs that may act as domain control entities (services and applications). The PCE working group is specifying extensions [PCEActive] that allow an active PCE to control, using PCEP, MPLS or GMPLS Label Switched Paths (LSPs), thus making it applicable for the CPSI for MPLS and GMPLS switches.

4.7. BFD

Bidirectional Forwarding Detection (BFD) [RFC5880] is an IETF-standardized network protocol designed for detecting path failures between two forwarding elements, including physical interfaces, subinterfaces, data link(s), and, to the extent possible, the forwarding engines themselves, with potentially very low latency. BFD can provide low-overhead failure detection on any kind of path between systems, including direct physical links, virtual circuits, tunnels, MPLS LSPs, multihop routed paths, and unidirectional links where there exists a return path as well. It is often implemented in some component of the forwarding engine of a system, in cases where the forwarding and control engines are separated.

With respect to Figure 1, a BFD agent can be implemented as a control-plane service or application that would use the CPSI towards the forwarding plane to send/receive BFD packets. However, a BFD agent is usually implemented as an application on the device and uses the forwarding plane to send/receive BFD packets and update the operational-plane resources accordingly. Services and applications of the control and management planes that monitor or have subscribed to changes of resources can learn about these changes through their respective interfaces and take any actions as necessary.

5. Summary

This document has been developed after a thorough and detailed analysis of related peer-reviewed literature, the RFC series, and documents produced by other relevant standards organizations. It has been reviewed publicly by the wider SDN community, and we hope that it can serve as a handy tool for network researchers, engineers, and practitioners in the years to come.

We conclude this document with a brief summary of the terminology of the SDN layer architecture. In general, we consider a network element as a composition of resources. Each network element has a forwarding plane (FP) that is responsible for handling packets in the data path and an operational plane (OP) that is responsible for managing the operational state of the device. Resources in the network element are abstracted by the Device and resource Abstraction Layer (DAL) to be controlled and managed by services or applications that belong to the control or management plane. The control plane (CP) is responsible for making decisions on how packets should be forwarded. The management plane (MP) is responsible for monitoring, configuring, and maintaining network devices. Service interfaces are abstracted by the Network Services Abstraction Layer (NSAL), where other network applications or services may use them. The taxonomy introduced in this document defines distinct SDN planes, abstraction layers, and interfaces; it aims to clarify SDN terminology and establish commonly accepted reference definitions across the SDN community, irrespective of specific implementation choices.

6. Security Considerations

This document does not propose a new network architecture or protocol and therefore does not have any impact on the security of the Internet. That said, security is paramount in networking; thus, it should be given full consideration when designing a network architecture or operational deployment. Security in SDN is discussed in the literature, for example, in [SDNSecurity], [SDNSecServ], and

[SDNSecOF]. Security considerations regarding specific interfaces (such as, for example, ForCES, I2RS, SNMP, or NETCONF) are addressed in their respective documents as well as in [RFC7149].

7. Informative References

[A4D05] Greenberg, A., Hjalmtysson, G., Maltz, D., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and H. Zhang, "A Clean Slate 4D Approach to Network Control and Management", ACM SIGCOMM Computer Communication Review, Volume 35, Issue 5, pp. 41-54, 2005.

[ALIEN] Parniewicz, D., Corin, R., Ogrodowczyk, L., Fard, M., Matias, J., Gerola, M., Fuentes, V., Toseef, U., Zaalouk, A., Belter, B., Jacob, E., and K. Pentikousis, "Design and Implementation of an OpenFlow Hardware Abstraction Layer", In Proceedings of the ACM SIGCOMM Workshop on Distributed Cloud Computing (DCC), Chicago, Illinois, USA, pp. 71-76, doi 10.1145/2627566.2627577, August 2014.

[Beacon] Erickson, D., "The Beacon OpenFlow Controller", In Proceedings of the second ACM SIGCOMM workshop on Hot Topics in Software Defined Networking, pp. 13-18, 2013.

[CAPBR] Brewer, E., "Towards Robust Distributed Systems", In Proceedings of the Symposium on Principles of Distributed Computing (PODC), 2000.

[CAPFN] Panda, A., Scott, C., Ghodsi, A., Koponen, T., and S. Shenker, "CAP for Networks", In Proceedings of the second ACM SIGCOMM workshop on Hot Topics in Software Defined Networking, pp. 91-96, 2013.

[CAPGL] Gilbert, S. and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services", ACM SIGACT News, Volume 33, Issue 2, pp. 51-59, 2002.

[CORBA] Object Management Group, "CORBA Version 3.3", November 2012, <http://www.omg.org/spec/CORBA/3.3/>.

[DIOPR] Denazis, S., Miki, K., Vicente, J., and A. Campbell, "Designing Interfaces for Open Programmable Routers", In "Active Networks", Springer Berlin Heidelberg, pp. 13-24, 1999.

[ESNet] Yu, J. and I. Al Ajarmeh, "An Empirical Study of the NETCONF Protocol", Sixth International Conference on Networking and Services, pp. 253-258, 2010.

[FCAPS] ITU, "Management Framework For Open Systems Interconnection (OSI) For CCITT Applications", ITU Recommendation X.700, September 1992, <http://www.itu.int/rec/T-REC-X.700-199209-I/en>.

[I2RSArch] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", Work in Progress, draft-ietf-i2rs-architecture-07, December 2014.

[I2RSInfo] Bahadur, N., Folkes, R., Kini, S., and J. Medved, "Routing Information Base Info Model", Work in Progress, draft-ietf-i2rs-rib-info-model-04, December 2014.

[I2RSProb] Atlas, A., Nadeau, T., and D. Ward, "Interface to the Routing System Problem Statement", Work in Progress, draft-ietf-i2rs-problem-statement-05, January 2015.

[ITUATM] ITU, "B-ISDN ATM Layer Specification", ITU Recommendation I.361, 1990, <http://www.itu.int/rec/T-REC-I.361-199902-I/en>.

[ITUSG11] ITU, "ITU-T Study Group 11: Protocols and test specifications", <http://www.itu.int/en/ITU-T/studygroups/2013-2016/11/Pages/default.aspx>.

[ITUSG13] ITU, "ITU-T Study Group 13: Future networks including cloud computing, mobile and next-generation networks", <http://www.itu.int/en/ITU-T/studygroups/2013-2016/13/Pages/default.aspx>.

[ITUSS7] ITU, "Introduction to CCITT Signalling System No. 7", ITU Recommendation Q.700, 1993, <http://www.itu.int/rec/T-REC-Q.700-199303-I/e>.

[ITUY3300] ITU, "Framework of software-defined networking", ITU Recommendation Y.3300, June 2014, <http://www.itu.int/rec/T-REC-Y.3300-201406-I/en>.

[KANDOO] Yeganeh, S. and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications", In Proceedings of the first ACM SIGCOMM workshop on Hot Topics in Software Defined Networks, pp. 19-24, 2012.

[NFVArch] ETSI, "Network Functions Virtualisation (NFV): Architectural Framework", ETSI GS NFV 002, October 2013, <http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf>.

[NOX] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and S. Shenker, "NOX: Towards an Operating System for Networks", ACM SIGCOMM Computer Communication Review, Volume 38, Issue 3, pp. 105-110, July 2008.

[NV09] Chowdhury, N. and R. Boutaba, "Network Virtualization: State of the Art and Research Challenges", Communications Magazine, IEEE, Volume 47, Issue 7, pp. 20-26, 2009.

[OF-CONFIG] Open Networking Foundation, "OpenFlow Management and Configuration Protocol (OF-Config 1.1.1)", March 2013, <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1-1-1.pdf>.

[OF08] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks", ACM SIGCOMM Computer Communication Review, Volume 38, Issue 2, pp. 69-74, 2008.

[ONFArch] Open Networking Foundation, "SDN Architecture, Version 1", June 2014, <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf>.

[OpenFlow] Open Networking Foundation, "The OpenFlow Switch Specification, Version 1.4.0", October 2013, <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>.

[P1520] Biswas, J., Lazar, A., Huard, J., Lim, K., Mahjoub, S., Pau, L., Suzuki, M., Torstensson, S., Wang, W., and S. Weinstein, "The IEEE P1520 standards initiative for programmable network interfaces", IEEE Communications Magazine, Volume 36, Issue 10, pp. 64-70, 1998.

[PCEActive] Crabbe, E., Minei, I., Medved, J., and R. Varga, "PCEP Extensions for Stateful PCE", Work in Progress, draft-ietf-pce-stateful-pce-10, October 2014.

[PENet] Hedstrom, B., Watwe, A., and S. Sakthidharan, "Protocol Efficiencies of NETCONF versus SNMP for Configuration Management Functions", Master's thesis, University of Colorado, 2011.

[PNSurvey99] Campbell, A., De Meer, H., Kounavis, M., Miki, K., Vicente, J., and D. Villela, "A Survey of Programmable Networks", ACM SIGCOMM Computer Communication Review, Volume 29, Issue 2, pp. 7-23, September 1992.

[Peregrine] Chiueh, D., Tu, C., Wang, Y., Wang, P., Li, K., and Y. Huang, "Peregrine: An All-Layer-2 Container Computer Network", In Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing, pp. 686-693, 2012.

[PiNA] Day, J., "Patterns in Network Architecture: A Return to Fundamentals", Prentice Hall, ISBN 0132252422, 2008.

[RCP] Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A., and J. van der Merwe, "Design and Implementation of a Routing Control Platform", In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation Volume 2, pp. 15-28, 2005.

[REST] Fielding, Roy, "Chapter 5: Representational State Transfer (REST)", in Disseration "Architectural Styles and the Design of Network-based Software Architectures", 2000.

[RFC0826] Plummer, D., "Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware", STD 37, RFC 826, November 1982, <http://www.rfc-editor.org/info/rfc826>.

[RFC1953] Newman, P., Edwards, W., Hinden, R., Hoffman, E., Ching Liaw, F., Lyon, T., and G. Minshall, "Ipsilon Flow Management Protocol Specification for IPv4 Version 1.0", RFC 1953, May 1996, <http://www.rfc-editor.org/info/rfc1953>.

[RFC2297] Newman, P., Edwards, W., Hinden, R., Hoffman, E., Liaw, F., Lyon, T., and G. Minshall, "Ipsilon's General Switch Management Protocol Specification Version 2.0", RFC 2297, March 1998, <http://www.rfc-editor.org/info/rfc2297>.

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999, <http://www.rfc-editor.org/info/rfc2578>.

[RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002, <http://www.rfc-editor.org/info/rfc3411>.

[RFC3412] Case, J., Harrington, D., Presuhn, R., and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3412, December 2002, <http://www.rfc-editor.org/info/rfc3412>.

[RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002, <http://www.rfc-editor.org/info/rfc3414>.

[RFC3417] Presuhn, R., "Transport Mappings for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3417, December 2002, <http://www.rfc-editor.org/info/rfc3417>.

[RFC3418] Presuhn, R., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002, <http://www.rfc-editor.org/info/rfc3418>.

[RFC3535] Schoenwaelder, J., "Overview of the 2002 IAB Network Management Workshop", RFC 3535, May 2003, <http://www.rfc-editor.org/info/rfc3535>.

[RFC3746] Yang, L., Dantu, R., Anderson, T., and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework", RFC 3746, April 2004, <http://www.rfc-editor.org/info/rfc3746>.

[RFC4271] Rekhter, Y., Li, T., and S. Hares, "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, January 2006, <http://www.rfc-editor.org/info/rfc4271>.

[RFC4655] Farrel, A., Vasseur, J., and J. Ash, "A Path Computation Element (PCE)-Based Architecture", RFC 4655, August 2006, <http://www.rfc-editor.org/info/rfc4655>.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, March 2009, <http://www.rfc-editor.org/info/rfc5424>.

[RFC5440] Vasseur, JP. and JL. Le Roux, "Path Computation Element (PCE) Communication Protocol (PCEP)", RFC 5440, March 2009, <http://www.rfc-editor.org/info/rfc5440>.

[RFC5531] Thurlow, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531, May 2009, <http://www.rfc-editor.org/info/rfc5531>.

[RFC5743] Falk, A., "Definition of an Internet Research Task Force (IRTF) Document Stream", RFC 5743, December 2009, <http://www.rfc-editor.org/info/rfc5743>.

[RFC5810] Doria, A., Hadi Salim, J., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and J. Halpern, "Forwarding and Control Element Separation (ForCES) Protocol Specification", RFC 5810, March 2010, <http://www.rfc-editor.org/info/rfc5810>.

[RFC5812] Halpern, J. and J. Hadi Salim, "Forwarding and Control Element Separation (ForCES) Forwarding Element Model", RFC 5812, March 2010, <http://www.rfc-editor.org/info/rfc5812>.

[RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, June 2010, <http://www.rfc-editor.org/info/rfc5880>.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC6632] Ersue, M. and B. Claise, "An Overview of the IETF Network Management Standards", RFC 6632, June 2012, <http://www.rfc-editor.org/info/rfc6632>.

[RFC7011] Claise, B., Trammell, B., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, September 2013, <http://www.rfc-editor.org/info/rfc7011>.

[RFC7047] Pfaff, B. and B. Davie, "The Open vSwitch Database Management Protocol", RFC 7047, December 2013, <http://www.rfc-editor.org/info/rfc7047>.

[RFC7149] Boucadair, M. and C. Jacquenet, "Software-Defined Networking: A Perspective from within a Service Provider Environment", RFC 7149, March 2014, <http://www.rfc-editor.org/info/rfc7149>.

[RFC7276] Mizrahi, T., Sprecher, N., Bellagamba, E., and Y. Weingarten, "An Overview of Operations, Administration, and Maintenance (OAM) Tools", RFC 7276, June 2014, <http://www.rfc-editor.org/info/rfc7276>.

[RINA] Day, J., Matta, I., and K. Mattar, "Networking is IPC: A Guiding Principle to a Better Internet", In Proceedings of the 2008 ACM CoNEXT Conference, Article No. 67, 2008.

[RouteFlow] Nascimento, M., Rothenberg, C., Salvador, M., Correa, C., de Lucena, S., and M. Magalhaes, "Virtual Routers as a Service: The RouteFlow Approach Leveraging Software-Defined Networks", In Proceedings of the 6th International Conference on Future Internet Technologies, pp. 34-37, 2011.

[SDNACS] Kreutz, D., Ramos, F., Verissimo, P., Rothenberg, C., Azodolmolky, S., and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey", Networking and Internet Architecture (cs.NI), arXiv:1406.0440, 2014.

[SDNHistory] Feamster, N., Rexford, J., and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks", ACM Queue, Volume 11, Issue 12, 2013.

[SDNNFV] Haleplidis, E., Hadi Salim, J., Denazis, S., and O. Koufopavlou, "Towards a Network Abstraction Model for SDN", Journal of Network and Systems Management: Special Issue on Management of Software Defined Networks, pp. 1-19, 2014.

[SDNSecOF] Kloti, R., Kotronis, V., and P. Smith, "OpenFlow: A Security Analysis", 21st IEEE International Conference on Network Protocols (ICNP) pp. 1-6, October 2013.

[SDNSecServ] Scott-Hayward, S., O'Callaghan, G., and S. Sezer, "SDN Security: A Survey", In IEEE SDN for Future Networks and Services (SDN4FNS), pp. 1-7, 2013.

[SDNSecurity] Kreutz, D., Ramos, F., and P. Verissimo, "Towards Secure and Dependable Software-Defined Networks", In Proceedings of the second ACM SIGCOMM workshop on Hot Topics in Software Defined Networking, pp. 55-60, 2013.

[SDNSurvey] Nunes, B., Mendonca, M., Nguyen, X., Obraczka, K., and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks", IEEE Communications Surveys and Tutorials, DOI:10.1109/SURV.2014.012214.00180, 2014.

[SLTSDN] Jarraya, Y., Madi, T., and M. Debbabi, "A Survey and a Layered Taxonomy of Software-Defined Networking", IEEE Communications Surveys and Tutorials, Volume 16, Issue 4, pp. 1955-1980, 2014.

[SoftRouter] Lakshman, T., Nandagopal, T., Ramjee, R., Sabnani, K., and T. Woo, "The SoftRouter Architecture", In Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networking, 2004.

[Tempest] Rooney, S., van der Merwe, J., Crosby, S., and I. Leslie, "The Tempest: A Framework for Safe, Resource Assured, Programmable Networks", Communications Magazine, IEEE, Volume 36, Issue 10, pp. 42-53, 1998.

Acknowledgements

The authors would like to acknowledge Salvatore Loreto and Sudhir Modali for their contributions in the initial discussion on the SDNRG mailing list as well as their document-specific comments; they helped put this document in a better shape.

Additionally, we would like to thank (in alphabetical order) Shivleela Arlimatti, Roland Bless, Scott Brim, Alan Clark, Luis Miguel Contreras Murillo, Tim Copley, Linda Dunbar, Ken Gray, Deniz Gurkan, Dave Hood, Georgios Karagiannis, Bhumip Khasnabish, Sriganesh Kini, Ramki Krishnan, Dirk Kutscher, Diego Lopez, Scott Mansfield, Pedro Martinez-Julia, David E. Mcdysan, Erik Nordmark, Carlos Pignataro, Robert Raszuk, Bless Roland, Francisco Javier Ros Munoz, Dimitri Staessens, Yaakov Stein, Eve Varma, Stuart Venters, Russ White, and Lee Young for their critical comments and discussions at IETF 88, IETF 89, and IETF 90 and on the SDNRG mailing list, which we took into consideration while revising this document.

We would also like to thank (in alphabetical order) Spencer Dawkins and Eliot Lear for their IRSG reviews, which further refined this document.

Finally, we thank Nobo Akiya for his review of the section on BFD, Julien Meuric for his review of the section on PCE, and Adrian Farrel and Benoit Claise for their IESG reviews of this document.

Kostas Pentikousis is supported by [ALIEN], a research project partially funded by the European Community under the Seventh Framework Program (grant agreement no. 317880). The views expressed here are those of the author only. The European Commission is not liable for any use that may be made of the information in this document.

Contributors

The authors would like to acknowledge (in alphabetical order) the following persons as contributors to this document. They all provided text, pointers, and comments that made this document more complete:

- Daniel King for providing text related to PCEP.
- Scott Mansfield for information regarding current ITU work on SDN.
- Yaakov Stein for providing text related to the CAP theorem and SDO-related information.
- Russ White for text suggestions on the definitions of control, management, and application.

Authors' Addresses

Evangelos Haleplidis (editor)
University of Patras
Department of Electrical and Computer Engineering
Patras 26500
Greece

EMail: ehalep@ece.upatras.gr

Kostas Pentikousis (editor)
EICT GmbH
Torgauer Strasse 12-15
10829 Berlin
Germany

EMail: k.pentikousis@eict.de

Spyros Denazis
University of Patras
Department of Electrical and Computer Engineering
Patras 26500
Greece

EMail: sdena@upatras.gr

Jamal Hadi Salim
Mojatatu Networks
Suite 400, 303 Moodie Dr.
Ottawa, Ontario K2H 9R4
Canada

EMail: hadi@mojatatu.com

David Meyer
Brocade

EMail: dmm@1-4-5.net

Odysseas Koufopavlou
University of Patras
Department of Electrical and Computer Engineering
Patras 26500
Greece

EMail: odysseas@ece.upatras.gr