

SDN Research Group
Internet-Draft
Intended status: Standards Track
Expires: August 13, 2014

LM. Contreras
Telefonica I+D
CJ. Bernardos
UC3M
February 9, 2014

Cooperating Layered Architecture for SDN
draft-contreras-sdnrg-layered-sdn-00

Abstract

The Software Defined Networking paradigm proposes the separation of the control plane from the data plane in the network nodes and its logical centralization on a control entity. All the network intelligence is moved to this central entity. Typically, such central entity is seen as a compendium of interacting control functions in a vertical, tight integrated fashion. The relocation of the control functions from a number of distributed network nodes to a logical central entity conceptually places together a number of control capabilities with different purposes. As a consequence, the existing solutions do not provide a clear separation between services and transport control.

This document describes a new proposal named Cooperating Layered Architecture for SDN. The idea behind that is to differentiate the control functions associated to transport from those related to services, in such a way that they can be provided and maintained independently, and can follow their own evolutionary way.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 13, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. Architecture overview	4
3.1. Functional strata	6
3.1.1. Transport stratum	6
3.1.2. Service stratum	6
3.1.3. Recursiveness	7
3.2. Plane separation	7
3.2.1. Control Plane	7
3.2.2. Management Plane	7
3.2.3. Resource Plane	7
4. Deployment scenarios	8
4.1. Full SDN environments	8
4.1.1. Multiple Service strata associated to a single Transport stratum	8
4.1.2. Single service stratum associated to multiple Transport strata	8
4.2. Hybrid environments	8
4.2.1. SDN Service stratum associated to a legacy Transport stratum	8
4.2.2. Legacy Service stratum associated to an SDN Transport stratum	9
5. IANA Considerations	9
6. Security Considerations	9
7. Normative References	9
Authors' Addresses	9

1. Introduction

Software Defined Networking (SDN) proposes the separation of the control plane from the data plane in the network nodes and its logical centralization on a control entity. A programmatic interface is defined between such entity and the network nodes, which functionality is now simplified to purely perform traffic forwarding. Through that interface, the central control entity instructs the nodes and modifies their traffic forwarding behavior.

All the intelligence is moved to such central entity. Typically, such central entity is seen as a compendium of interacting control functions in a vertical, tight integrated fashion.

This approach presents a number of issues:

- o Unclear responsibilities between actors involved in a service provision and delivery.
- o Complex reuse of functions for the provision of services.
- o Closed, monolithic control architectures.
- o Difficult interoperability and interchangeability of functional components.
- o Blurred business boundaries among providers.

The relocation of the control functions from a number of distributed network nodes to a logical central entity conceptually places together a number of control capabilities with different purposes. As a consequence, the existing solutions do not provide a clear separation between services and transport control.

This document describes a new proposal named Cooperating Layered Architecture for SDN (CLAS). The idea behind that is to differentiate the control functions associated to transport from those related to services, in such a way that they can be provided and maintained independently, and can follow their own evolutionary way.

Despite such differentiation it is required a close cooperation between service and transport layers and associated components to provide an efficient usage of the resources.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 [RFC2119].

Additionally, the following acronyms are used in this document.

CLAS: Cooperating Layered Architecture for SDN

SDN: Software Defined Networking

SLA: Service Level Agreement

3. Architecture overview

Current operator networks support multiple services (e.g., mobile, fixed, enterprise, etc) on a variety of transport technologies. The provision and delivery of a service independently of the underlying transport capabilities requires a separation of the service related functionalities and, ideally, an abstraction of the transport network to hide the particularities of each technology while offering a common set of capabilities.

Such separation can provide configuration flexibility and adaptability either from the point of view of the services or the transport network. Multiple services can be provided on top of a common transport network, and similarly, different technologies can support a certain service. A close coordination among them is required for a consistent service delivery.

An example of that could be the guarantee of some Quality of Service (QoS) level. Different QoS offerings could be present at both service and transport layers. Vertical mechanisms for linking both service and transport QoS mechanisms should be in place to provide the quality guarantees to the end user.

This document presents a proposal called Cooperating Layered Architecture for SDN (CLAS). In this architecture the logically centralized control functions are separated in two blocks or layers. One of the layers comprises the service-related functions, whereas the other one contains the transport-related functions. The cooperation between the two layers is considered to be implemented through open, standard interfaces.

Figure 1 shows the CLAS architecture. It is based on functional separation in the NGN architecture defined by the ITU-T in [Y.2011]. Two strata of functionality are defined, namely the Service Stratum,

comprising the service-related functions, and the Transport Stratum, covering the transport ones. The functions on each of these layers are further grouped on control, management and user (or data) planes.

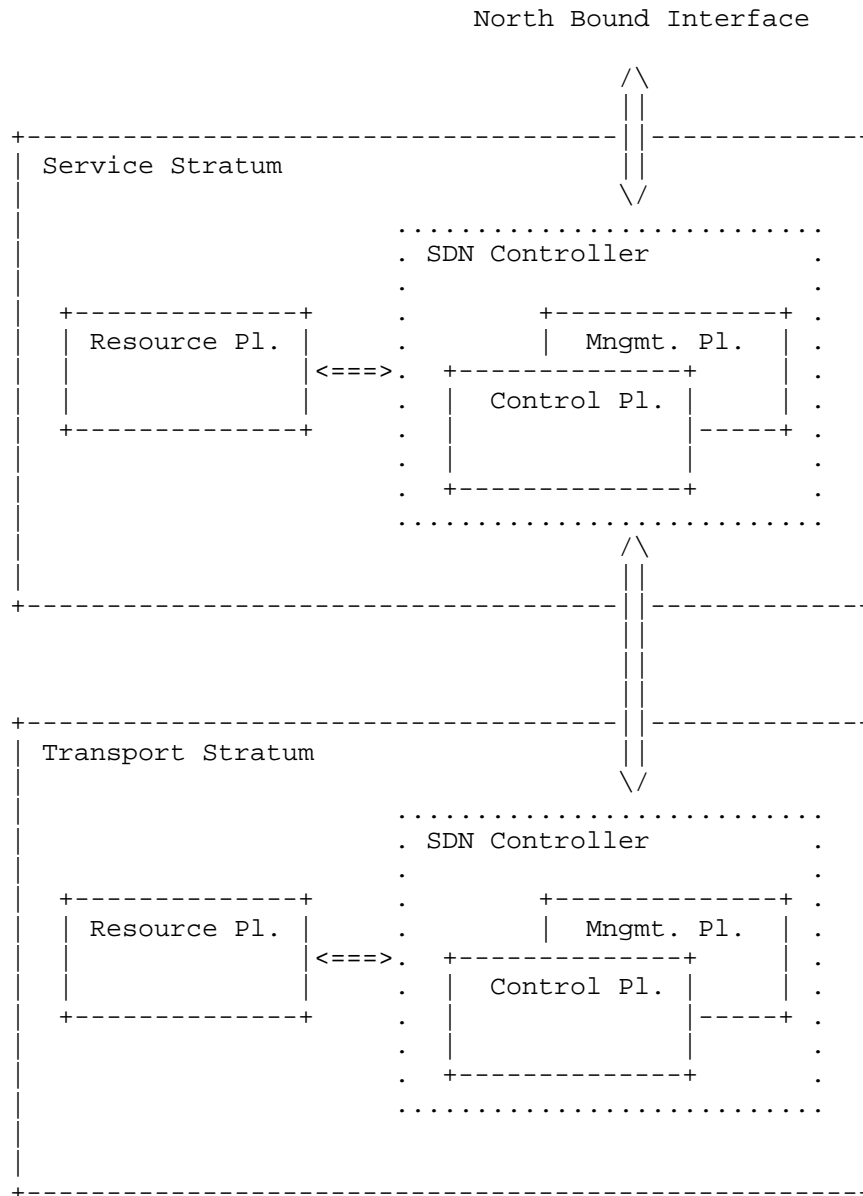


Figure 1: Cooperating Layered Architecture for SDN

In the CLAS architecture both the control and management functions are the ones logically centralized in an SDN controller, in such a way that separated SDN controllers are present in the Service and Transport strata. Furthermore, the generic user or data plane functions included in the NGN architecture are referred here as resource plane functions. The resource plane in each stratum is controlled by the corresponding SDN controller through an standard interface.

The SDN controllers cooperate for the provision and delivery of services. There is a hierarchy in which the Service SDN controller requests transport capabilities to the Transport SDN controller. Furthermore, the Transport SDN controller interacts with the Service SDN controller to inform it about events in the transport network that can motivate actions in the service layer.

The Service SDN controller acts as a client of the Transport SDN controller.

Despite it is not shown in the figure, the Resource planes of each stratum could be connected. This will depend on the kind of service provided. Furthermore, the Service stratum could offer a North Bound Interface towards external applications to expose network service capabilities to those applications.

3.1. Functional strata

As described before, the functional split separates transport-related functions from service-related functions. Both strata cooperate for a consistent service delivery.

3.1.1. Transport stratum

The Transport stratum comprises the functions focused on the pure transfer of end user data between the communication end points. The data forwarding nodes are part of the Resource plane. These nodes are controlled and managed by the Transport SDN controller. The Control plane in the SDN controller is in charge of instructing the forwarding devices to build the end to end data path for each communication. Finally, the Management plane performs management functions on those devices, like fault or performance management, as part of the Transport stratum capabilities.

3.1.2. Service stratum

The Service stratum contains the functions related to the provision of services and the capabilities offered to external applications. The Resource plane consists of the resources involved in the service

delivery, such as computing resources, registries, databases, etc. The Control plane is in charge of controlling and configuring those resources, as well as interacting with the Control plane of the Transport stratum in client mode for requesting transport capabilities for a given service. In the same way, the Management plane implements management actions on the service-related resources and interacts with the Management plane in the Transport stratum for a cooperating management between layers.

3.1.3. Recursiveness

Recursive layering can happen in some usage scenarios in which the Transport Stratum is itself structured in Service and Transport Stratum. This could be the case of the provision of a transport services complemented with advanced capabilities additional to the pure data transport (e.g., maintenance of a given SLA).

3.2. Plane separation

The CLAS architecture leverages on the SDN proposition of plane separation. As mentioned before, three different planes are considered for each stratum. The communication among these three planes (and with the corresponding plane in other strata) is based on open, standard interfaces.

3.2.1. Control Plane

The Control plane logically centralizes the control functions of each stratum and directly controls the corresponding resources. This plane is part of an SDN controller, and can interact with other control planes in the same or different strata for accomplishing control functions.

3.2.2. Management Plane

The Management plane logically centralizes the management functions for each stratum, including the management of the Control and Resource planes. This plane is also part of the SDN controller, and can interact with the corresponding management planes residing in SDN controllers of the same or different strata.

3.2.3. Resource Plane

The Resource plane comprises the resources for either the transport or the service functions. In some cases the service resources can be connected to the transport ones (e.g., being the terminating points of a transport function) whereas in other cases it can be decoupled

from the transport resources (e.g., one database keeping some register for the end user).

4. Deployment scenarios

Different situations can be found depending on the characteristics of the networks involved in a given deployment.

4.1. Full SDN environments

This case considers the fact that the networks involved in the provision and delivery of a given service have SDN capabilities.

4.1.1. Multiple Service strata associated to a single Transport stratum

A single Transport stratum can provide transfer functions to more than one Service strata. The Transport stratum offers a standard interface to each of the Service strata. The Service strata are the clients of the Transport stratum. Some of the capabilities offered by the Transport stratum can be isolation of the transport resources, independent routing, etc.

4.1.2. Single service stratum associated to multiple Transport strata

A single Service stratum can make use of different Transport strata for the provision of a certain service. The Service stratum interfaces each of the Transport strata with standard protocols, and orchestrates the provided transfer capabilities for building the end to end transport needs.

4.2. Hybrid environments

This case considers scenarios where one of the strata is legacy totally or in part.

4.2.1. SDN Service stratum associated to a legacy Transport stratum

An SDN service stratum can interact with a legacy Transport stratum through some interworking function able to adapt SDN-based control and management service-related commands to legacy transport-related protocols, as expected by the legacy Transport stratum. The SDN controller in the Service stratum is not aware of the legacy nature of the underlying Transport stratum.

4.2.2. Legacy Service stratum associated to an SDN Transport stratum

A legacy Service stratum can work with an SDN-enabled Transport stratum through the mediation of an interworking function capable to interpret commands from the legacy service functions and translate them into SDN protocols for operating with the SDN-enabled Transport stratum.

5. IANA Considerations

TBD.

6. Security Considerations

TBD. Security in the communication between strata to be addressed.

7. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [Y.2011] "General principles and general reference model for Next Generation Networks", ITU-T Recommendation Y.2011, October 2004.

Authors' Addresses

Luis M. Contreras
Telefonica I+D
Ronda de la Comunicacion, s/n
Sur-3 building, 3rd floor
Madrid 28050
Spain

Email: lmcm@tid.es
URI: <http://people.tid.es/LuisM.Contreras/>

Carlos J. Bernardos
Universidad Carlos III de Madrid
Av. Universidad, 30
Leganes, Madrid 28911
Spain

Phone: +34 91624 6236
Email: cjbc@it.uc3m.es
URI: <http://www.it.uc3m.es/cjbc/>

SDNRG
Internet-Draft
Intended status: Informational
Expires: January 5, 2015

E. Haleplidis
S. Denazis
University of Patras
K. Pentikousis
EICT
J. Hadi Salim
Mojatatu Networks
D. Meyer
Brocade
O. Koufopavlou
University of Patras
July 4, 2014

SDN Layers and Architecture Terminology
draft-haleplidis-sdnrg-layer-terminology-05

Abstract

Software-Defined Networking (SDN) can in general be defined as a new approach for network programmability. Network programmability refers to the capacity to initialize, control, change, and manage network behavior dynamically via open interfaces as opposed to relying on closed-box solutions and proprietary-defined 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 is the layer structure in an SDN architecture and how do layers interface with each other. This document aims to answer these questions and provide a concise reference document for SDNRG, in particular, and the SDN community, in general, based on relevant peer-reviewed literature and documents in the RFC series.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 5, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. SDN Layers and Architecture	6
3.1. Overview	7
3.2. Network Devices	11
3.3. Control Plane	11
3.4. Management Plane	13
3.5. Control vs Management plane	14
3.5.1. Timescale	14
3.5.2. Ephemeral/Persistent	14
3.5.3. Locality	14
3.5.4. CAP Theorem	15
3.6. Network Services Abstraction Layer	16
3.7. Application Plane	17
4. SDN Model View	17
4.1. ForCES	17
4.2. NETCONF	18
4.3. OpenFlow	19
4.4. I2RS	20
4.5. BFD	21
4.6. SNMP	21
5. Acknowledgements	22
6. IANA Considerations	22
7. Security Considerations	22
8. Informative References	22
Authors' Addresses	28

1. Introduction

Software-Defined Networking (SDN) is a relevant new term for 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]. Another view of what SDN is defined in [RFC7149] as a set of techniques used to facilitate the design, the delivery and the 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 the 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].

Feamster et al. [SDNHistory] review the historical evolution of the programmable networks research area, starting with earlier efforts which date back to the 1980s. As the authors document, many of the ideas, concepts and concerns are applicable to the latest R&D in SDN, and SDN standardization we may add, 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 data 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 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 in IETF and the research community. Of particular interest, for example, is to determine which layers comprise the SDN architecture and which interfaces and their corresponding attributes are best suitable 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 document which reflects current approaches regarding the SDN layers 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 aims to address the potential work item in the SDNRG charter named "Survey of SDN approaches and Taxonomies", fostering better understanding of prominent SDN technologies in a technology-impartial and business-agnostic manner. As such, we do not make any value statements nor discuss the applicability of any of the frameworks examined in this draft for any particular purpose. Instead, we document their characteristics and attributes and classify them, thus providing a taxonomy. Already there are a number of survey papers regarding SDN that discuss taxonomies such as [SLTSDN] and [SDNACS].

This document does not constitute a new IETF standard nor a new specification, and aims to receive rough consensus within SDNRG to be published in the IRTF Stream as per [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 with 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 component, physical or virtual, available within a system. Resources can be very simple or fine-grained, e.g. a port, 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. In case the entities are not in the same physical location, the interface is usually implemented as a network protocol. In case the entities are collocated in the same physical location the interface can be a protocol or an open/proprietary software inter-process communication Application Programming Interface (API).

Application (App) - A piece of software that utilizes underlying services to perform a function. Application operation can be parametrized, 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 network device part responsible for forwarding traffic.

Operational Plane (OP) - The network device part responsible for managing the overall device operation.

Control Plane (CP) - Part of the network functionality that is assigned to control one or more network devices. CP instructs network devices with respect to how to treat and forward packets. The control plane interacts primarily with the forwarding plane and less with the operational plane.

Management Plane (MP) - Part of the network functionality responsible for monitoring, configuring and maintaining one or more network devices. The management plane is mostly related with the operational plane and less with the forwarding plane.

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.

3. SDN Layers and Architecture

Figure 1 provides a detailed high-level overview of the current SDN architecture abstractions. 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 calls 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.

The concept of separation via IPCs is explored in RINA [RINA] where the premise is that all network communications is considered an IPC and that allows a recursive approach on creating hierarchical network connections. RINA's [RINA] approach has a lot of commonalities with the described SDN layer abstractions as we can also view these layers as being hierarchical stacked on top of each other as needed.

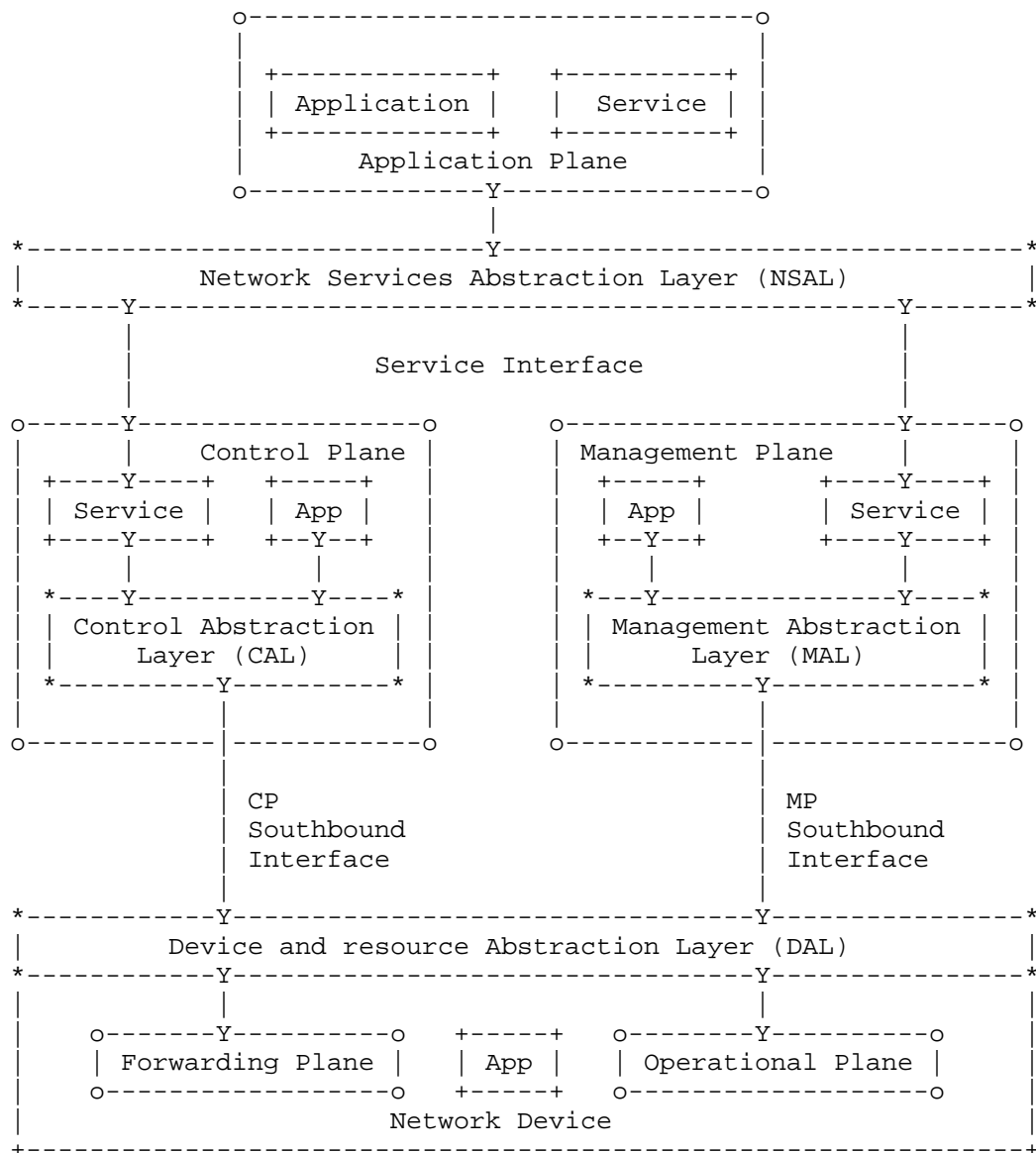


Figure 1: SDN Layer Architecture

3.1. Overview

This document follows a network device centric approach: Control refers to the device packet handling capability, while Management refers to the overall device operation aspects. We view a network

device as a complex resource which 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 device.

The reader should keep in mind throughout this document that we make no distinction between "physical" and "virtual" resources, 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 on whether a resource is implemented as an overlay or as a part/component of some other device. Finally, network device software can run on so-called "bare metal" or on a virtualized substrate.

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:

- o Forwarding Plane - Responsible for handling packets in the datapath. 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.
- o Operational Plane - Responsible for managing the operational state of the Network Device, e.g. active/inactive, number of ports, port status, etc. The Operational Plane is usually the termination point for management plane services and applications. The operational plane relates to (operational aspects of) Network Device resources such as ports, memory, and so on. Members of the SDNRG have different opinions in regards to the operational plane, with the argument being that it does not constitute a plane by itself but an amalgamation of functions in the data plane. For others it is considered a plane similar to the rest of the planes as a plane may be considered as a distinction between different areas of operations.
- o Control Plane - Responsible for taking decisions on how packets should be forwarded by one or more Network Devices and pushing such decisions down to the Network Devices to be executed. 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 example, 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.

- o Management Plane - Responsible for monitoring, configuring and maintaining network devices, e.g. taking 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.
- o Application Plane - The plane where applications that rely on the network to provide services for end users and processes 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.

All planes mentioned above are connected via Interfaces (as 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 co-located 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 plane and, thus, be able to use both the CPSI and MPSI. 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 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 provide through NSAL services to other applications or services. Additional examples include virtual resources that are realized on top of a physical resources and hierarchical control plane controllers [KANDOO].

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 tend to place the management plane on top of the control plane, which may be interpreted as having the control plane acting as a service to the management plane. Traditionally, the control plane was tightly coupled with the device. When taken as whole, the control plane was distributed network-wide. On the other hand, the management plane has been traditionally centralized and was 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 device's forwarding and operational plane resources to the control and management plane. 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 CP southbound interface and the DAL from the applications and services of the Control Plane.

The Management Abstraction Layer (MAL) abstracts the MP 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.

We observe that the view presented in this document is quite well-aligned with recently published work by the ONF; see [ONFArch]. A key difference, however, is that the ONF architecture does not include the management plane in its scope.

SDN related activities have begun in many other SDO's, such as:

ITU: Architectural work [ITUSG13] but have not been published at the time this document was written.

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) and 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.

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 both 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 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 ForCES [RFC5812] and OpenFlow [OpenFlow]. Examples of the Operational Plane abstraction model include the ForCES model [RFC5812], the YANG model [RFC6020] and SNMP MIBs [RFC3418].

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.

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 planes, colloquially referred to as the "east-west" interface, is usually implemented through gateway protocols like BGP [RFC4271]. However, the corresponding protocol messages are in fact 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:

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

The CPSI is usually defined with the following characteristics:

- o time-critical interface which requires low latency and sometimes high bandwidth in order to perform many operations in short order.
- o 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 interprocess 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 CPSIs.

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, BGP, etc.

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, which consists of network devices, is running optimally by communicating with the network devices's 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] [RFC3535] usually include:

- o Fault and Monitoring management
- o Configuration management

Normally MSPI, in contrast to the CPSI, is not a time-critical interface and does not share the CPSI requirements.

MSPI is [RFC3535] typically closer to human interaction than the control plane and therefore the MSPI usually has the following characteristics:

- o It is oriented more towards usability, with optimal wire performance being a secondary concern.
- o 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], as mentioned in [RFC6632], where 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 require more processing capabilities to parse.

The MPSI can range from a protocol, to an API or even interprocess communication. If the Management Plane is not embedded in the network device, the MSPI is certainly a protocol. Examples of MPSIs are ForCES [RFC5810], NETCONF [RFC6241], 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 and fault detection and recovery applications.

Management Plane Services provide access to other services or applications above the Management Plane.

3.5. Control vs Management plane

During the SDNRG meetings as well via the list, one of the most commonly discussed topic, in regards to this document, was the clear distinction between control and management. We have identified the following characteristics that together or each one may provide the necessary differentiator between the planes.

3.5.1. Timescale

Timescale refers to how fast an application in the respective plane react or need to manipulate the forwarding or operational plane of the device. In general, the control plane needs to send updates very often within the range of milliseconds and that requires a high bandwidth and low latency links. In contrast the management plane reacts generally at very slow timeframes, minutes, hours or even days, e.g. in the case of changing the configuration state of the device, or and thus do not need to be very efficient on the wire.

3.5.2. Ephemeral/Persistent

Another distinction that discussed was the distinction between ephemeral versus persistent state. Ephemeral state is state that may have a very limited lifespan, such as routing decisions, and thus is usually associated with the control plane. On the other hand, persistent state is state that may have a larger and extended lifespan which may range from hours to days and months and is usually associated with the management plane. Persistent state is also usually associated with data store of the state.

3.5.3. Locality

Before the concept of centralizing the controller, usually the control plane is local to device and distributed whilst the management plane is usually centralized and remote from the device. However, as has been noted before, centralizing, or "locally centralizing" the controller tends to muddle the distinction of the control and management plane on locality.

3.5.4. CAP Theorem

An additional distinction was introduced in the 89th IETF and refers to the CAP theorem.

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 and 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 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 called the CAP theorem

Correctly forwarding a packet through a network, is a computational problem. One of the major abstractions that SDN posits - all network elements are computational resources that perform the single 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, control plane operations are usually local and fast (available), while management plane operations are usually centralized (consistent) and slow.

The CAP theorem provides insights on SDN performance. For example, in regards to locality, although not explicitly stated, modern SDN philosophy, centralizing the controller, stresses consistency. The controller acts as a consistent global database, and specific mechanisms ensure that a packet entering the network is handled consistently by all switches and acts like a management entity. The

issue of tolerance to loss of connectivity to the controller is not addressed by the basic SDN model. When an SDN switch can't 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 improves partition tolerance, but at the cost of loss of absolute consistency

3.6. Network Services Abstraction Layer

The Network Services Abstraction Layer (NSAL) provides access from services of the control, management and application planes to services and applications of the application plane. 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 prefixed it with "Network" 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 or proprietary protocols such as NETCONF, inter-process communications, CORBA interfaces, etc.

Two leading standards of service interface are RESTful interfaces and RPC interfaces. Both follow a client-server architecture and use XML or JSON to pass messages but each have some slightly different characteristics.

RESTful interfaces, designed with 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, not through a fixed interface.

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

Individual procedures are identified using an identifier

A client needs to know the procedure name and the 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 CSPI and MSPI.

The SDN northbound interface is implemented in the Network Services Abstraction Layer of Figure 1.

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

4.1. ForCES

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

The ForCES model 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 and secondly with a protocol to transport information.

ForCES models the Forwarding Plane using Logical Functional Blocks (LFBs) which are connected in a graph, composing the Forwarding Element (FE). LFBs are described in an XML language, based on an XML schema.

LFB definitions include:

- o Base and custom-defined datatypes
- o Metadata definitions
- o Input and Output ports
- o Operational parameters, or components
- o Capabilities
- o Event definitions

The ForCES model can be used to define LFBs from fine- to coarse-grained as needed irrelevant 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). ForCES is a protocol designed for high throughput and fast updates.

ForCES [RFC5810] can be mapped to the framework illustrated in Figure 1 as follows:

- o The ForCES model can be used to describe DAL, both for the Operational and the Forwarding Plane, using LFBs.
- o The ForCES protocol can then be both the CPSI and the MPSI. ForCES is inherently specified for the CPSI and satisfies its requirements, however it can also be utilized for the MPSI.
- o CAL and MAL must be able to utilize the ForCES protocol.

4.2. NETCONF

The Network Configuration Protocol (NETCONF [RFC6241]), is an IETF-standardized 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 an Extensible Markup Language (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 NETCONF, 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 datatype and allows custom defined datatypes as well.

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

NETCONF can be mapped to the framework illustrated in Figure 1 as follows:

- o The YANG model [RFC6020] is suitable for specifying DAL for the operational plane and NETCONF [RFC6241] for the MPSI.
- o Technically, the YANG model [RFC6020] can be used to specify DAL for the Forwarding plane as well. That said, in principle NETCONF [RFC6241] is a management protocol 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 by Standford, and currently under active standards development through the Open Networking Foundation. Initially, the goal was to provide a way for researchers to run experimental protocols in a production network [OFSIGC]. OpenFlow provides a protocol with which a controller may manage a static model of an OpenFlow switch.

An OpenFlow switch consists of one or more flow tables which perform packet lookups, actions on a success packet lookup and forwarding, a group table and an OpenFlow channel to an external controller. The switch communicates with the controller which manages the switch via the OpenFlow protocol.

OpenFlow has undergone many revisions. The current version is 1.4 [OpenFlow] and supports amongst others, multiple controllers for high availability and extensible flow match field protocol messages to support arbitrary match fields. Efforts to define OpenFlow 2.0 [PPIPP] are already underway aiming to provide an abstract forwarding model to provide protocol independence and device programmability.

OpenFlow can be mapped to the framework illustrated in Figure 1 as follows:

- o The Openflow switch specifications [OpenFlow] covers DAL for the Forwarding Plane and provides the specification for CPSI.
- o The OF-CONFIG protocol [OF-CONFIG] based on the YANG model [RFC6020], provides DAL for the Operational Plane 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.
- o CAL must be able to utilize the OpenFlow protocol.
- o MAL must be able to utilize the NETCONF protocol.

4.4. I2RS

I2RS is currently developed by a recently-established IETF working group. The intention is to provide 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, I2RS aims to make the routing information base (RIB) programmable thus enabling new kinds of network provisioning and operation.

I2RS does 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 [I-D.ietf-i2rs-problem-statement] discusses previously-defined IETF protocols and data models such as ForCES, YANG, NETCONF, and SNMP.

Currently the I2RS working group is developing an Information Model [I-D.ietf-i2rs-rib-info-model] in regards to the Network Services Abstraction Layer for the I2RS agent.

I2RS can be mapped to the framework illustrated in Figure 1 as follows:

- o The I2RS architecture [I-D.ietf-i2rs-architecture] encompasses the Control and Application Planes and uses any CPSI and DAL that is available, whether that may be ForCES, OpenFlow or another Interface.
- o The I2RS agent is a Control Plane Service. All services or applications on top of that belong to either the Control, Management or the 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 provide access even to other than control applications.

4.5. BFD

Bidirectional Forwarding Detection (BFD) [RFC5880], is an IETF network protocol designed for detecting communication failures between two forwarding elements which are directly connected. It is intended to be implemented in some component of the forwarding engine of a system, in cases where the forwarding and control engines are separated.

BFD provides low-overhead detection of faults even on physical media that do not support failure detection of any kind, such as Ethernet, virtual circuits, tunnels and MPLS Label Switched Paths.

BFD could be mapped to the framework illustrated in Figure 1 either as:

1. A control plane service or application that would use the CPSI towards the forwarding plane to send/receive BFD packets.
2. Or, better, as it was intended for, i.e. as an application that runs on the device itself and uses the forwarding plane to send/receive BFD packets and update the operational plane resources accordingly.

4.6. SNMP

The Simple Network Management Protocol (SNMP) is an IETF management protocol and is currently at the third version called SNMPv3 and described in STD 62, RFC 3417 [RFC3417], RFC 3412 [RFC3412] and RFC 3414 [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 (OID). 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 SMIV2 [RFC2578]

SNMP could be mapped to the framework illustrated in Figure 1 as:

1. SNMP MIBs are usually used to describe DAL for the Operational Plane.
2. SNMP is usually used for the MPSI.

5. Acknowledgements

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

Additionally the authors would like to acknowledge Russ White, Linda Dunbar, Robert Raszuk, Pedro Martinez-Julia, Lee Young, Yaakov Stein, Shivleela Arlimatti, Gurkan Deniz, Scott Brim, Carlos Pignataro, Ramki Krishnan, Bless Roland, Tim Copley, Francisco Javier Ros Munoz, Sriganesh Kini, Alan Clark, Erik Nordmark, Scott Mansfield, Dirk Kutscher, Roland Bless, David E Mcdysan, Bhumip Khasnabish and Georgios Karagiannis for their critical comments and discussions at the IETF 88 and 89 meetings (and the SDNRG mailing list), which we took into consideration while revising this document.

Special thanks to Yaakov Stein for providing text related to the CAP theorem and Scott Mansfield for information regarding ITU status on SDN

6. IANA Considerations

This memo makes no requests to IANA.

7. Security Considerations

TBD

8. Informative References

- [A4D05] Greenberg, Albert, et al., "A clean slate 4D approach to network control and management", ACM SIGCOMM Computer Communication Review 35.5 (2005): 41-54 , 2005.
- [CAPBR] Eric A. Brewer, "Towards robust distributed systems.", Symposium on Principles of Distributed Computing (PODC). 2000 , 2000.

- [CAPGL] Seth Gilbert, and Nancy Ann Lynch., "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", ACM SIGACT News 33.2 (2002): 51-59. , 2002.
- [DIOPR] Denazis, Spyros, Kazuho Miki, John Vicente, and Andrew Campbell., "Designing interfaces for open programmable routers.", In Active Networks, pp. 13-24. Springer Berlin Heidelberg, 1999 , 1999.
- [ESNet] Yu, James, and Imad Al Ajarmeh., "An empirical study of the NETCONF protocol.", In Networking and Services (ICNS), 2010 Sixth International Conference on, pp. 253-258. IEEE, 2010. , 2010.
- [FCAPS] International Telecommunication Union, "X.700: Management Framework For Open Systems Interconnection (OSI) For CCITT Applications", September 1992,
<<http://www.itu.int/rec/T-REC-X.700-199209-I/en>>.
- [I-D.ietf-i2rs-architecture]
Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", draft-ietf-i2rs-architecture-04 (work in progress), June 2014.
- [I-D.ietf-i2rs-problem-statement]
Atlas, A., Nadeau, T., and D. Ward, "Interface to the Routing System Problem Statement", draft-ietf-i2rs-problem-statement-04 (work in progress), June 2014.
- [I-D.ietf-i2rs-rib-info-model]
Bahadur, N., Folkes, R., Kini, S., and J. Medved, "Routing Information Base Info Model", draft-ietf-i2rs-rib-info-model-03 (work in progress), May 2014.
- [ITUATM] CCITT, Geneva, Switzerland, "CCITT Recommendation 1.361, B-ISDN ATM Layer Specification", 1990.
- [ITUSG13] Telecommunication Standardization sector of ITU, "ITU, Study group 13", 2013, <<http://www.itu.int/en/ITU-T/studygroups/2013-2016/13/Pages/default.aspx>>.
- [ITUSS7] Telecommunication Standardization sector of ITU, "ITU, Q.700 : Introduction to CCITT Signalling System No. 7", 1993.

- [KANDOO] Hassas Yeganeh, Soheil, and Yashar Ganjali., "Kandoo: a framework for efficient and scalable offloading of control applications.", In Proceedings of the first workshop on Hot topics in software defined networks, pp. 19-24. ACM SIGCOMM, 2012. , 2012.
- [NV09] Chowdhury, NM Mosharaf Kabir, and Raouf Boutaba, "Network virtualization: state of the art and research challenges", Communications Magazine, IEEE 47.7 (2009): 20-26 , 2009.
- [OF-CONFIG] Open Networking Foundation, "OpenFlow Management and Configuration Protocol 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, Nick, et al., "OpenFlow: enabling innovation in campus networks", ACM SIGCOMM Computer Communication Review 38.2 (2008): 69-74 , 2008.
- [OFSIGC] McKeown, Nick, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner., "OpenFlow: enabling innovation in campus networks.", ACM SIGCOMM Computer Communication Review 38, no. 2 (2008): 69-74. , 1998.
- [ONFArch] Open Networking Foundation, "SDN Architecture Overview", December 2013,
<<https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf>>.
- [OpenFlow] Open Networking Foundation, "The OpenFlow 1.4 Specification.", October 2013,
<<https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>>.
- [P1520] Biswas, Jit, Aurel A. Lazar, J-F. Huard, Koonseng Lim, Semir Mahjoub, L-F. Pau, Masaaki Suzuki, Soren Torstensson, Weiguo Wang, and Stephen Weinstein., "The IEEE P1520 standards initiative for programmable network interfaces.", Communications Magazine, IEEE 36, no. 10 (1998): 64-70. , 1998.

- [PENet] Hedstrom, Brian, Akshay Watwe, and Siddharth Sakthidharan, "Protocol Efficiencies of NETCONF versus SNMP for Configuration Management Functions", PhD dissertation, Master's thesis, University of Colorado, 2011 , 2011.
- [PNSurvey99] Campbell, Andrew T., et al, "A survey of programmable networks", ACM SIGCOMM Computer Communication Review 29.2 (1999): 7-23 , September 1992.
- [PPIPP] Bosshart, Pat, Dan Daly, Martin Izzard, Nick McKeown, Jennifer Rexford, Dan Talayco, Amin Vahdat, George Varghese, and David Walker., "Programming Protocol-Independent Packet Processors.", arXiv preprint arXiv:1312.1719 (2013). , 2013.
- [RCP] Caesar, Matthew, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus 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. USENIX Association, 2005. , 2005.
- [REST] Fielding, Roy, "Fielding Dissertation: Chapter 5: Representational State Transfer (REST).", 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.
- [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.
- [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.
- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

- [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.
- [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.
- [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.
- [RFC3417] Presuhn, R., "Transport Mappings for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3417, December 2002.
- [RFC3418] Presuhn, R., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.
- [RFC3535] Schoenwaelder, J., "Overview of the 2002 IAB Network Management Workshop", RFC 3535, May 2003.
- [RFC4271] Rekhter, Y., Li, T., and S. Hares, "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, January 2006.
- [RFC5531] Thurlow, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531, May 2009.
- [RFC5743] Falk, A., "Definition of an Internet Research Task Force (IRTF) Document Stream", RFC 5743, December 2009.
- [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.
- [RFC5812] Halpern, J. and J. Hadi Salim, "Forwarding and Control Element Separation (ForCES) Forwarding Element Model", RFC 5812, March 2010.
- [RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, June 2010.

- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.
- [RFC6632] Ersue, M. and B. Claise, "An Overview of the IETF Network Management Standards", RFC 6632, June 2012.
- [RFC7047] Pfaff, B. and B. Davie, "The Open vSwitch Database Management Protocol", RFC 7047, December 2013.
- [RFC7149] Boucadair, M. and C. Jacquenet, "Software-Defined Networking: A Perspective from within a Service Provider Environment", RFC 7149, March 2014.
- [RINA] John Day, Ibrahim Matta, and Karim Mattar., "Networking is IPC: a guiding principle to a better internet.", In Proceedings of the 2008 ACM CoNEXT Conference, p. 67. ACM, 2008. , 2008.
- [RouteFlow] Nascimento, Marcelo R., Christian E. Rothenberg, Marcos R. Salvador, Carlos NA Correa, Sidney C. de Lucena, and Mauricio F. 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. ACM, 2011. , 2011.
- [SDNACS] Diego Kreutz, Fernando M. V. Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, Steve Uhlig, "Software-Defined Networking: A Comprehensive Survey.", arXiv preprint arXiv:1406.0440 , 2014.
- [SDNHistory] Feamster, Nick, Jennifer Rexford, and Ellen Zegura., "The Road to SDN", ACM Queue11, no. 12 (2013): 20. , 2013.
- [SLTSDN] Yosr Jarraya, Taous Madi, and Mourad Debbabi, "A Survey and a Layered Taxonomy of Software-Defined Networking", To be published in Communications Surveys and Tutorials, IEEE Issue: 99 , 2014.

[SoftRouter]

Lakshman, T. V., T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo., "The softrouter architecture.", In Proc. ACM SIGCOMM Workshop on Hot Topics in Networking. 2004. , 2004.

[Tempest]

Rooney, Sean, Jacobus E. van der Merwe, Simon A. Crosby, and Ian M. Leslie., "The Tempest: a framework for safe, resource assured, programmable networks.", Communications Magazine, IEEE 36, no. 10 (1998): 42-53 , 1998.

Authors' Addresses

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

Email: ehalep@ece.upatras.gr

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

Email: sdena@upatras.gr

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

Email: k.pentikousis@eict.de

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

SDNRG
Internet-Draft
Intended status: Standards Track
Expires: January 5, 2015

Y. Xia, Ed.
S. Jiang, Ed.
T. Zhou, Ed.
S. Hares
Huawei Technologies Co., Ltd
July 4, 2014

NEMO (NETwork MOdeling) Language
draft-xia-sdnrg-nemo-language-00

Abstract

The North-Bound Interface (NBI), located between the network control plane and the applications, is essential to enable the application innovations and nourish the eco-system of SDN.

While most of the NBIs are provided in the form of API, this document proposes the NETwork MOdeling (NEMO) language which is another NBI fashion. Concept, model and syntax are introduced in the document.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 5, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	4
3. Related work	4
4. The NEMO Language overview	5
4.1. Network Model of the NEMO Language	5
4.2. Primitives	7
5. The NEMO Language Examples	9
6. Security Considerations	10
7. IANA Considerations	10
8. Acknowledgements	10
9. Informative References	10
Authors' Addresses	11

1. Introduction

While SDN (Software Defined Network) is becoming one of the most important directions of network evolution, the essence of SDN is to make the network more flexible and easy to use. The North-Bound Interface (NBI), located between the control plane and the applications, is essential to enable the application innovations and nourish the eco-system of SDN by abstracting the network capabilities/information and opening the abstract/logic network to applications.

The NBI is usually provided in the form of API (Application Programming Interface). Different vendors provide self-defined API sets. Each API set, such as OnePK from Cisco and OPS from Huawei, often contains hundreds of specific APIs. Diverse APIs without consistent style are hard to remember and use, and nearly impossible to be standardized.

Most of those APIs are designed by network domain experts, who are used to thinking from the network system perspective. The interface designer does not know how the users will use the device and exposes information details as much as possible. It enables better control of devices, but leaves huge burden of selecting useful information to users without well training. Since the NBI is used by network users, a more appropriate design is to think from the user perspective and abstract the network from the top down.

[I-D.sdng-service-description-language] describe the requirements for a service description language and the design considerations.

A top-down NBI design contains following features:

- o Express user intent

To simplify the operation, applications or users can use the NBI directly to describe their requirements for the network without taking care of the implementation. All the parameters without user concern will be concealed by the NBI.

- o Platform independent

With the NBI, the application or user can describe network demand in a generic way, so that any platform or system can get the identical knowledge and consequently execute to the same result. Any low-level and device/vendor specific configurations and dependencies should be avoided.

- o Intuitive domain specific language (DSL) for network

The expression of the DSL should be human-friendly and be easily understood by network operators. DSL should be directly used by the system.

- o Privilege control

Every application or user is authorized within a specific network domain, which can be physical or virtual. While different network domains are isolated without impact, the application or user may have access to all the resource and capabilities within its domain. The user perception of the network does not have to be the same as the network operators. The proposed Network Modeling (NEMO) language works on the user's view so the users can create topologies based on the resources the network-operators allow them to have.

- o Declarative style

As described above, NEMO language is designed to help defining service requirement to network, detailed configurations and instructions performed by network devices are opaque to network operators. So NEMO language should be declarative rather than imperative.

To implement such an NBI design, we can learn from the successful case of SQL (Structured Query Language), which simplified the complicated data operation to a unified and intuitive way in the form of language. Applications do not care about the way of data storage and data operation, but to describe the demand for the data storage

and operation and then get the result. As a data domain DSL, SQL is simple and intuitive, and can be embedded in applications. So what we need for the network NBI is a set of "network domain SQL".

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

Network service also "service" for short, is the service logic that contains network operation requirements;

Network APP also "APP" for short, is the application to implement the network service;

Network user also "user" for short, is the network administrator or operator.

3. Related work

YANG is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF), NETCONF remote procedure calls, and NETCONF notifications [RFC6020]. Although it is extensible for more data modeling in addition to NETCONF, YANG is not capable of describing high level network requirements, such as SLA (Service Level Agreement). YANG is designed for north-bound interfaces of the device, which is also the south-bound of the controller. It is not proper to model the north-bound interface of the controller, aka the NBI. Moreover, the YANG is not capable of describing the service processing logic, which typically includes transition of conditions and states.

UML (Unified Modeling Language) is a powerful modeling language, which is domain agnostic. It is hard to describe the network demand, and cannot be embedded in network applications. UML is appropriate to describe the model behind the NBI language not the NBI itself.

With the emergence of the SDN concept, it is a consensus to simplify the network operation, which leads to many cutting-edge explorations in the academic area.

Nick McKeown from Stanford University proposed the SFNet [TSFNet], which translated the high level network demand to the underlying

controller interfaces. By concealing the low level network details, the controller simplified the operation of resource, flow, and information for applications. The SFNet is used for the SDN architecture design, and does not go into the NBI design.

Jennifer from Princeton University designed the Frenetic [Frenetic] based on the OpenFlow protocol. It is an advanced language for flow programming, and systematically defines the operating model and mode for the flow. However, the network requirement from the service is not only the flow operations, but also includes operations of resource, service conditions, and service logic.

In the book [PBNM], John Strassner defined the policy concept and proposed the formal description for network operations by using the policy. The method for querying network information is absent in the book. Virtual tenant network and operations to the tenant network are not considered.

All these investigations direct to the future SDN that use simple and intuitive interfaces to describe the network demands without complex programming.

4. The NEMO Language overview

NEMO language is a domain specific language (DSL) based on abstraction of network models and conclusion of operation patterns. It provides NBI fashion in the form of language. With limited number of key words and expressions, NEMO language defines the entity and capability models for users with different view of network abstraction, and enables network users/applications to describe their demands for network resources, services and logical operations in an intuitive expression. And finally the NEMO language description can be explained and executed by a language engine.

4.1. Network Model of the NEMO Language

Behind the NEMO language, there is a set of meta-models abstracting the network demands from the top down according to the service requirement. Those demands can be divided into two types: the demand for network resources and the demands for network behaviors.

The network resource is composed of three kinds of entities: node, link and flow. Each entity contains property and statistic information. With a globally unique identifier, the network entity is the basic object for operation.

- o Node model: describes the entity with the capability of packet processing. According to the functionality, there are three types of node
 - * The forwarding node (FN) only deals with L2/3 forwarding. It forwards packets according to the forwarding table and modifies packet heads.
 - * The processing node (PN) provides L4-7 network services, and will modify the body of packets.
 - * The logical node (LN) describes a set of network elements and their links, such as subnet, autonomous system, and internet. It conceals the internal topology and exposes properties as one entity. It also enables iteration, i.e., a network entity may include other network entities.
- o Link model: describes the connectivity between node entities. According to the forwarding capability, links are usually divided into layer 2 and layer 3 types
- o Flow model: describes a sequence of packets with certain common characters, such as source/destination IP address, port, and protocol. From the northbound perspective, flow is the special traffic with user concern, which may be per device or across many devices. So the flow characters also include ingress/egress node, and so on.

Network behavior includes the information and control operations.

The information operation provides two methods to get the network information for users.

- o Query: a synchronous mode to get the information, i.e., one can get the response when a request is sent out.
- o Notification: an asynchronous mode to get the information, i.e., with one request, one or multiple responses will be sent to the subscriber automatically whenever trigger conditions meet.

The NEMO language uses policy to describe the control operation.

- o Policy: control the behavior of specific entities by APP, such as flow policy, node policy. All the policies follow the same pattern "with <condition>, to execute <action>", and can be applied to any entity

4.2. Primitives

The primitives of NEMO language are derived from the network model, and fall into four categories.

a. Resource access primitives

```
Node/UnNode  entity_id  Type {FN|PN|LN}
                Owner node_id
                Properties key1 ,value1
```

Node/UnNode: create/delete a node
 Entity id: system allocated URI for the node entity
 Type: Node type of FN (forwarding node), PN (processing node) or LN (logical node)
 Owner: since the node can be nested, this primitive figures out which node the new one belongs to
 Properties: other properties to describe the node in the form of (key, value).

```
Link/UnLink  entity_id  Endnodes (node1_id,node2_id)
                SLA key,value
                Properties key1 ,value1
```

Link/UnLink: create/delete a link.
 Entity id: system allocated URI for the link entity
 Endnodes: two end-node IDs of the link
 SLA: SLA description for the link
 Properties: other properties to describe the link in the form of (key, value).

```
Flow/UnFlow  entity_id  Match/UnMatch key1, value1|
                Range(value, value) |
                Mask(value, value)
                Properties key1 ,value1
```

Flow/UnFlow: create/delete a flow.
 Match/UnMatch: create/delete match items for the flow
 Range: describe the range of the value
 Mask: use mask to describe a range of the value
 Properties: other properties to describe the flow in the form of (key, value).

b. Behavior primitives

```

Query    key  Value {value}
          From entity_id

```

```

Query:      generate a synchronously query
key:        the parameter name to be queried
Value:      the return value for the query
From:       the entity to be queried (define entity_id).

```

```

Policy/UnPolicy  policy_id  Appliesto  entity_id
                  Condition {expression}
                  Action {"forwardto"|"drop"|"gothrough"|"
                        "bypass"|"guaranteeSLA"|"Set"|"
                        "Packetout"|"Node|UnNode|Link|Unlink}

```

```

Policy/UnPolicy: create/delete a policy
Appliesto:       apply the policy to an entity
Condition:       condition to execute the policy
Action:          actions to be executed when conditions are met

```

```

Notification/UnNotification  entity_id  On  key
                              Every  period
                              RegisterListener
                              callbackfunc

```

```

Notification/UnNotification: create/delete a notification for an
                             entity
On:                          the notification will monitor the state change of a
                             parameter identified by the "key"
Every:                       time period at which to report the state
RegisterListener: the callback function that is used to process the
                             notification.

```

c. Connection management primitives

```

Connect  conn_id      Address  ip_address
          Port  port_num
Disconnect  conn_id

```

```

Connect:      set up a connection to the controller
Address:      IP address of the controller to connect to
Port:         port of the controller to connect to
Disconnect:   disconnect to the controller.

```

d. Transaction primitives

Transaction
Commit

Transaction: indicate the beginning of a transaction
Commit: commit to execute the transaction

5. The NEMO Language Examples

A tenant needs two connections to carry different service flows between two datacenters.

one connection of the tenant is 40G bandwidth with less than 400ms delay, another connection is 100M bandwidth with less than 50ms delay.

```
{
  Link Link1_id
    Endnodes (DC1_node_id, DC2_node_id)
    Property "NAME", "DC1_DC2_link_one", "Bandwith", 40G, "Delay", 400ms
  Link Link2_id
    Endnodes (DC1_node_id, DC2_node_id)
    Property "NAME", "DC1_DC2_link_two", "Bandwith", 100M, "Delay", 50ms
}
```

The tenant has two types of traffic, CDN sync traffic uses high bandwidth connection and online game traffic uses low latency connection.

```
{
  Flow flow1_id
    Match "srcip", "10.0.1.1/24", "dstip", "20.0.1.1/24", "Port", "55555"
    Property "NAME", "CDN sync flow", "Bidirection", "true"
  Flow flow2_id
    Match "srcip", "10.0.1.1/24", "dstip", "20.0.1.1/24", "Port", "56663"
    Property "NAME", "online Game", "Bidirection", "true"
  Policy policy1_id
    Appliesto flow1_id
    Action "forwardto", link1_id
  Policy policy2_id
    Appliesto flow2_id
    Action "gothrough", link2_id
}
```

The tenant wants the online game traffic to go through WOC in nighttime before it is carried by low latency connection.

```
{
  Policy policy3_id
    Appliesto flow2_id
    Condition {Time>18:00 or Time< 2:00}
    Action "gothrough",{woc_node_id ,link2_id}
}
```

6. Security Considerations

Because the network customers are allowed to customize their own services, they may bring potentially big impacts to a running IP network. A strong user authentication mechanism is needed for the northbound interface of the SDN controller. User authorization should be carefully managed by the network administrator to avoid any dangerous operations and prevent any abuse of network resources.

7. IANA Considerations

This memo includes no request to IANA.

8. Acknowledgements

The authors would like to thanks the valuable comments made by Wei Cao, Xiaofei Xu, Fuyou Miao and Wenyang Lei.

This document was produced using the xml2rfc tool [RFC2629].

9. Informative References

[Frenetic]

Foster, N., Harrison, R., Freedman, M., Monsanto, C., Rexford, J., Story, A., and D. Walker, "Frenetic: A Network Programming Languages, ICFP' 11", .

[I-D.sdnrng-service-description-language]

Xia, Y., Jiang, S., and S. Hares, "Requirements for a Service Description Language and Design Considerations, draft-xia-sdnrg-service-description-language-00, Work in progress", July 2014.

[PBNM]

Strassner, J., "Policy-Based Network Management: Solutions for the Next Generation, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.", 2003.

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.
- [TSFNet] Yap, K., Huang, T., Dodson, B., Lam, M., and N. McKeown, "Towards Software-Friendly Networks, APSys 2010, pp:49-54, 2010, New Delhi, India.", .

Authors' Addresses

Yinben Xia (editor)
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: xiayinben@huawei.com

Sheng Jiang (editor)
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: jiangsheng@huawei.com

Tianran Zhou (editor)
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: zhoutianran@huawei.com

Susan Hares
Huawei Technologies Co., Ltd
7453 Hickory Hill
Saline, CA 48176
USA

Email: shares@ndzh.com

SDNRG
Internet-Draft
Intended status: Informational
Expires: January 5, 2015

Y. Xia, Ed.
S. Jiang, Ed.
S. Hares
Huawei Technologies Co., Ltd
July 4, 2014

Requirements for a Service Description Language and Design
Considerations
draft-xia-sdnrg-service-description-language-00

Abstract

The more and more complicated IP networks require a new interaction mechanism between their customers and their networks. A service description language is needed to enable customers to easily describe their diverse service requirements. SDN controller would compile these service requirements into device configurations. This document analyzes requirements for such service description language and gives considerations for designing such service description language.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 5, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Analysis of Network Customer's Service Demands	4
3.1. An Example of Service Requirements	4
4. Design Consideration	4
4.1. A Description Example of Service Requirements	5
5. Security Considerations	6
6. IANA Considerations	6
7. Acknowledgements	6
8. Informative References	7
Authors' Addresses	7

1. Introduction

The IP networks of the Internet Service Providers (ISPs) and data centers are becoming more and more big and complicated. Simultaneously, the services that are demanded by their customers, particularly the upper layer applications, are also becoming more and more complicated. The rigid service models are lacking the flexibility to meet the various requirements/scenarios. A better form would be that the network customers are allowed to customize their own services as they need.

Recently, there are many efforts have been made on opening the IP devices and networks. Today, there are many open APIs from different vendors, such as OnePK from Cisco, OPS from Huawei, and etc. They are mainly device-oriented interfaces. Interface to the Routing System (I2RS) WG is working to allow information, policies, and operational parameters to be injected into and retrieved from the routing system. It makes possible for user application to directly intervene in the running routes, or deploy specific demands.

However, such open interfaces are bottom-up designed according to the devices. One has to be very familiar with devices in order to correctly "programming" his demands. Such interfaces are far from user-friendly. Particularly, for many upper layer applications, their demands may involve hundreds and thousands devices. It is very difficult for a network customer to direct program network devices.

Software-Defined Networking (SDN) controller has taken such responsibility: hide the complexity of networks from customers,

receive abstracted service demands from customers, and compile/translate the demands into detailed control operations that can directly execute on network devices. This would allow network customers to be released from the burden of selecting useful information and capability from vast information and capability of the infrastructure network.

The interactions between SDN controller and network customers should be as simple as possible. The network customers should be allowed to describe their demands in their own way, which is as close as possible to their nature demands. Consequently, the northbound interface of SDN controller must be different from the northbound interface of network devices, which actually matches the southbound interface of SDN controller. This northbound interface of SDN controller should be designed using top-domain methodology, so that network customers can use it as easy as possible.

This document starts from analyzing the demands from network customers, tries to epurate technical requirements for a service description language and the design consideration for a such language. A few typical examples of network customers' demands and their description examples are also given.

The interaction between the SDN controller and the IP infrastructure network, such as how the information and capability of infrastructure networks are abstracted, how network capabilities are executed and how the service logic is translated, are out of scope of this document.

2. Terminology

SDN Controller An application in Software-Defined Networking (SDN) that manages flow control. It controls manages a number of network devices in the infrastructure network, regarding how to forwarding IP packets.

Northbound Interface of SDN Controller An interactive interface between SDN controller and network customers. It receives the customer orders in both data form or service logic form.

Northbound Interface of Network Device An interactive interface that allows SDN controller, or network management system to directly operate the network devices.

Service Description Language A language used to describe specific service demands by the network customers.

3. Analysis of Network Customer's Service Demands

The network customers do not care the detailed configurations of each device, or flow entries in each device, even when their service flows go through these devices. They do not want to be bored the detailed device-oriented operations, such as tunnel management, isolation with other services, PBR configurations of different devices. What the network customers care about is the service demand they require and the service quality they receive.

3.1. An Example of Service Requirements

A typical network customer's demand would firstly start from connectivities: connect the two datacenters that locate in two cities. For security reasons, the customer normally want to organize all their connectivities as a virtual network. {add the example of link}

Then, the customer normally need to appoint the quality of service or choose from certain Service Level Agreement (SLA) for this connectivity.

Typically, traffics of customers could be categorized into several classes, which match with different SLAs.

Furthermore, the customer may demand some flows go through a certain intermedia server, such as firewall.

The customer may want to organize his few demands together with certain choosing circumstances

In some scenarios, the customer flows may be needed to be presented by various form. For example, client/server flows are normally come from different and distributed sources.

4. Design Consideration

The purpose of a service description language is to describe the network customer's requirements. The SDN controller or network administration system then compile them into the operations of network devices.

The language should have the below ability:

- o be able to describe customer traffics which can be identified as flows;

- o be able to describe access node, virtual network, servers, and other network entities that the network customers apperceive;
- o be able to describe QoS, SLAs and other relevant properties;
- o be able to describe logic that combines a few demands together with certain choosing circumstances;
- o be able to describe the necessary information from the network (e.g. SDN controller), so that the network customer can describe their requirements accordingly;
- o easy to extend in order to support various new services or demands that may appear in the future.

4.1. A Description Example of Service Requirements

A tenant needs two connections to carry different service flows between two datacenters. one connection of the tenant is 40G bandwidth with less than 400ms delay, another connection is 100M bandwidth with less than 50ms delay.

```
{  
  Link Link1_id  
    Endnodes (DC1_node_id, DC2_node_id)  
    Property "NAME","DC1_DC2_link_one","Bandwith",40G, "Delay",400ms  
  
  Link Link2_id  
    Endnodes (DC1_node_id, DC2_node_id)  
    Property "NAME","DC1_DC2_link_two","Bandwith",100M, "Delay",50ms  
}
```

The tenant has two types of traffic, CDN sync traffic uses high bandwidth connection and online game traffic uses low latency connection.

```
{
  Flow flow1_id
    Match "srcip","10.0.1.1/24","dstip",?20.0.1.1/24,"Port","55555"
    Property "NAME", "CDN sync flow", "Bidirection","true?"

  Flow flow2_id
    Match "srcip","10.0.1.1/24","dstip",?20.0.1.1/24,"Port","56663"
    Property "NAME", "online Game", "Bidirection","true?"

  Policy policy1_id
    Appliesto flow1_id
    Action "forwardto", link1_id

  Policy policy2_id
    Appliesto flow2_id
    Action "gothrough", link2_id
}
```

the tenant wants the online game traffic to go through WOC in nighttime before it is carried by low latency connection.

```
{
  Policy policy3_id
    Appliesto flow2_id
    Condition {Time>18:00 or Time< 2:00}
    Action "gothrough", {woc_node_id ,link2_id}
}
```

5. Security Considerations

Because the network customers are allowed to customize their own services, they may bring potentially big impacts to a running IP network. A strong user authentication mechanism is needed for the northbound interface of the SDN controller. User authorization should be carefully managed by the network administrator to avoid any dangerous operations and prevent any abuse of network resources.

6. IANA Considerations

This memo includes no request to IANA.

7. Acknowledgements

The authors would like to thanks the valuable comments made by Wei Cao, Xiaofei Xu, Fuyou Miao and Wenyang Lei.

This document was produced using the xml2rfc tool [RFC2629].

8. Informative References

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

Authors' Addresses

Yinben Xia (editor)
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: xiayinben@huawei.com

Sheng Jiang (editor)
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: jiangsheng@huawei.com

Susan Hares
Huawei Technologies Co., Ltd
7453 Hickory Hill
Saline, CA 48176
USA

Email: shares@ndzh.com