SDNRG
Internet-Draft
Intended status: Informational
Expires: September 4, 2014

E. Haleplidis
S. Denazis
University of Patras
K. Pentikousis
EICT
J. Hadi Salim
Mojatatu Networks
D. Meyer
Brocade
O. Koufopavlou
University of Patras
March 3, 2014

SDN Layers and Architecture Terminology
draft-haleplidis-sdnrg-layer-terminology-04

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 propietary-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

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 September 4, 2014.

Copyright Notice

Table of Contents

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 [I-D.sin-sdnrg-sdn-approach] 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 accomodate legacy networking solutions in their (then new) programmable environment. Further, the concept of separating the control and data planes, which is promiment 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.

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 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.

```
                   o--------------------------------o
                   |                                |
                   | +-------------+   +----------+ |
                   | | Application |   |  Service | |
                   | +-------------+   +----------+ |
                   |       Application Plane        |
                   o---------------Y----------------o
                                   |
     *-----------------------------Y---------------------------------*
     |           Network Services Abstraction Layer (NSAL)           |
     *------Y------------------------------------------------Y-------*
            |                                                |
            |               Service Interface                |
            |                                                |
     o------Y------------------o       o---------------------Y------o
     |      |    Control Plane |       | Management Plane    |      |
     | +----Y----+   +-----+   |       |  +-----+       +----Y----+ |
     | | Service |   | App |   |       |  | App |       | Service | |
     | +----Y----+   +--Y--+   |       |  +--Y--+       +----Y----+ |
     |      |           |      |       |     |               |      |
     | *----Y-----------Y----* |       | *---Y---------------Y----* |
     | | Control Abstraction | |       | | Management Abstraction | |
     | |     Layer (CAL)     | |       | |      Layer (MAL)       | |
     | *----------Y----------* |       | *----------Y-------------* |
     |            |            |       |            |               |
     o------------|------------o       o------------|---------------o
                  |                                 |
                  | CP                              | MP
                  | Southbound                      | Southbound
                  | Interface                       | Interface
                  |                                 |
     *------------Y---------------------------------Y----------------*
     |         Device and resource Abstraction Layer (DAL)           |
     *------------Y---------------------------------Y----------------*
     |            |                                 |                |
     |    o-------Y----------o   +-----+   o--------Y----------o     |
     |    | Forwarding Plane |   | App |   | Operational Plane |     |
     |    o------------------o   +-----+   o-------------------o     |
     |                       Network Device                          |
     +---------------------------------------------------------------+
```

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.

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 finetune 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 hierachical 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, respectively. Variations of DAL may abstract both planes or either of the two.

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. Architectural work has also begun in ITU [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 a physical or virtual network element. As mentioned above, this document makes no distinction between these. 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 can be exposed via a Device and resource Abstraction Layer (DAL), which may comprise 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

- 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:

- It is oriented more towards usability, with optimal wire performance being a secondary concern.

- Messages tend to be less frequent than in the CPSI

As an example of usability versus performance, we refer to the consensus of the 2002 IAB Workshop [RFC3535], 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. 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

3.6. 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 arbitraty 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.

- 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:

- 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.

- 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.

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 for their critical comments and discussions at the IETF 88 meeting (and the SDNRG mailing list), which we took into consideration while revising this document.

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.

[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-02 (work in progress), February 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-00 (work in progress), August 2013.

[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-02 (work in progress), February 2014.

[I-D.sin-sdnrg-sdn-approach]
Boucadair, M. and C. Jacquenet, "Software-Defined Networking: A Perspective From Within A Service Provider", draft-sin-sdnrg-sdn-approach-09 (work in progress), January 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.

[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.

[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.

[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.

[SDNHistory]
Feamster, Nick, Jennifer Rexford, and Ellen Zegura., "The Road to SDN", ACM Queue11, no. 12 (2013): 20. , 2013.

[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