SCTE ISBE CABLE-TEC EXPO '16
SEPTEMBER 26-29 PHILADELPHIA

SCTE Society of Cable Telecommunications Engineers
ISBE International Society of Broadband Experts

A PRACTICAL GUIDE TO IMPLEMENTING SOFTWARE DEFINED DOCSIS

A Technical Paper prepared for SCTE/ISBE by

David S Early, Ph.D.
Chief Architect and Data Scientist
Applied Broadband
2741 Mapleton Ave, Boulder, CO 80304
720-259-5621
david@appliedbroadband.com

Table of Contents

List of Figures

Introduction

SDN carries the promise of enhanced and expanded services with a more manageable operational environment. While SDN technology is maturing, state-of-the-art is focused on monolithic controller architectures for traditional networks, with all functions and control being highly centralized – and subject to the scaling and computational limits of a single server architectural approach.

This paper presents a modular approach to implementing SDN, leveraging existing monolithic controller architectures for distributed management, but implementing a new centralized control plane that encompasses only those elements necessary for overarching systems management. Distributed monolithic controllers are relatively autonomous, while centralized management platforms give the MSO centralized control of the entire network. This agile new modular approach allows for the immediate leveraging of existing SDN management platforms without the limitations associated with a purely monolithic approach.

This paper provides an on overview of a practical SDN implementation of PacketCable Multimedia (PCMM), along with real-world considerations for SDN architectures and technologies when applied to Cable Broadband networks. A demonstration will be provided to illustrate.

SDN: From Promise to Delivery

1. Making the Case for SDN in DOCSIS

The use of a software-defined network (SDN) in broadband means resources can be virtualized and allocated to whatever application or service requires it, when it needs it.

Though advantages to SDN-based architectures and applications have been proposed [1], the specific benefits when applied to a programmable DOCSIS [2][3] policy environment are as follows:

1. System and network resource elasticity. Traditional network and service provisioning software for DOCSIS networks has historically been burdened, by rigid system and network resource deployment constraints and difficult resource dimensioning limitations. The virtualization of DOCSIS low-level services using contemporary SDN architectures enables greater design flexibility in networks of scale while affording discrete upgrades to capacities at time of need.

2. A common platform across network applications. Capital, operational, and organizational economies of scale result when using multiple applications that access the same network services and related resources in a common way.

3. Service delivery agility. As a consequence (benefit) of both (1) and (2) above, new subscriber services that benefit from QoS treatment in the access network can quickly be supported.

Business agility is vital for the very survival of broadband organizations hence the ability to implement new services quickly is vital.

Traditional DOCSIS network architectures and services lack the flexibility required to support an agile business and spend more budget than is necessary to over-provision the network for peak utilization. A forward looking SDN approach is required for any organization looking to become agile, brings the dynamic features to the network, and gets maximum benefit from the network virtualization functions for a minimum overall cost. Virtualization has largely transformed the data center with flexible and automated server provisioning, but networking and storage infrastructure have not kept pace.

2. Myth vs Truth: Is SDN Really a Panacea for All Networking Ills?

While the SDN & OpenDaylight (ODL) [4] technology may still be a little new (and the hype a bit on the high side), there are real use cases for SDN and the OpenDaylight PCMM plugin in broadband networks – and we'll be surprised if products offering this and in-house build systems don't start taking advantage.

To start, an SDN controller in a software-defined network is the "brains" of the network, and it can be open or proprietary. The SDN controller is the strategic control point in the SDN network, relaying information to the southbound APIs (switches/routers) and the applications and business login via northbound APIs. Southbound interfaces (NETCONF and SNMP) and Protocol plugins (PCMM) facilitate efficient control over the network and enable the SDN controller to dynamically make changes according to real-time demands and needs.

The Network Configuration Protocol (NETCONF) defined by IETF RFC 6241 provides mechanisms to install, manipulate and delete the configuration of network devices. The Simple Network Management Protocol (SNMP), defined by IETF RFC 3411, allows collection and organization of information about managed devices on IP networks and modification of that information to change device behavior.

SDN allows routers and switches to match packets on multiple header fields, not just destination IP address. It allows a network controller to control entire networks for the single program, possibly even including a remote autonomous system as opposed to just immediate neighbors. SDN provides mechanism for direct control over packet handling (rather than indirect control via various routing protocols) and it offers the ability to perform many different actions on packets beyond simply just forwarding them. When you think about where it might make sense to deploy SDN, ISPs are the natural place to start. SDN controllers can benefit tens to hundreds of providers and it is possible for ISPs to adopt SDN without providers deploying new network equipment.

One significant challenge in utilizing existing SDN development work is the differences between "traditional" networks and DOCSIS networks. Much of the SDN world has focused on traditional networking applications, and the use of such protocols as OpenFlow [5]. When comparing OpenFlow to Cable access network technologies such as DOCSIS, fundamental differences exist:

- Topology: The underpinning HFC network is a physical tree/branch topology. The CMTS (master) and CM (slave) topology and resulting protocol design are fundamentally different than port-to-port topology of switched networks (e.g. Ethernet).
- Scale: The number of managed devices (entities) in a single managed DOCSIS network is well into the tens of millions and expected to grow further.
- Complexity: The Information model representing a complete DOCSIS network describes objects and attributes counted in the thousands.

DOCSIS, SDN and OpenDaylight

Experience has shown repeatedly that a monolithic approach to management systems has serious drawbacks. One very powerful pain point is consolidation of features into a single platform that would require monolithic upgrades for single features, monolithic management and monolithic failure potential.

In this section we address the SDN reference architecture and some proposed adjustments and enhancements that will help put SDN in context for broadband networks.

3. Theory

The SDN architecture abstraction presented by the ITU-T in [6] describes three functional layers and two interfaces between the layers (illustrated in Figure 1):

1. Application Layer: SDN and/or business applications that specify network behavior.
2. SDN Control Layer: "a means to dynamically and deterministically control the behavior of network resources…as instructed by the application layer" [6].
3. Resource Layer: Network devices and element management systems.

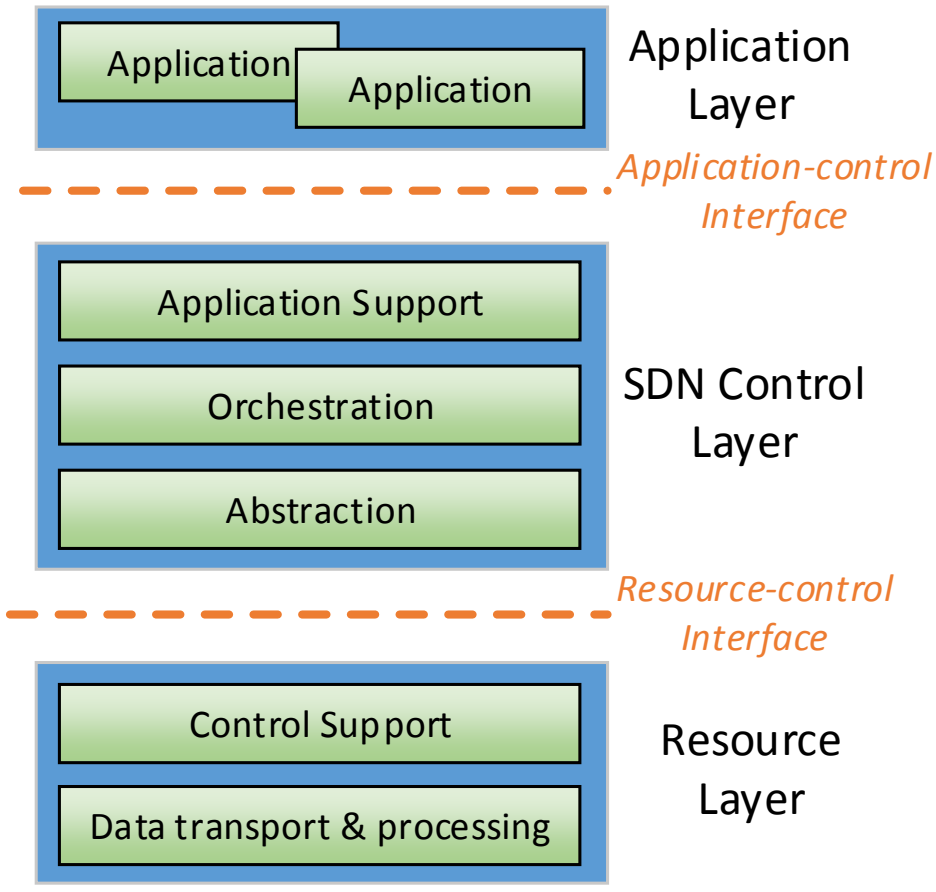

Figure 1 - SDN High Level Abstraction (ITU-T Y.3300 [6])

The two interfaces (Application-Control and Resource-Control interfaces) represent bidirectional communication between the components of each layer.

The current reality for PCMM is that SDN "controllers" perform very little if any autonomous modification to the requests that the Application Layer makes to the core. The Application Layer is implicitly responsible for making decisions about the nature of the request and the controller does not modify requests. The controller is mostly a router of requests to network resources.

4. Practice

Strong adherence to a theoretical layered architecture may or may not provide the functionality and architectural flexibility necessary for long-term evolution. We propose a modified and modular architecture that will leverage aspects of existing SDN controller platforms while maximizing current and future options for the core.

The modified architecture is shown in Figure 2.

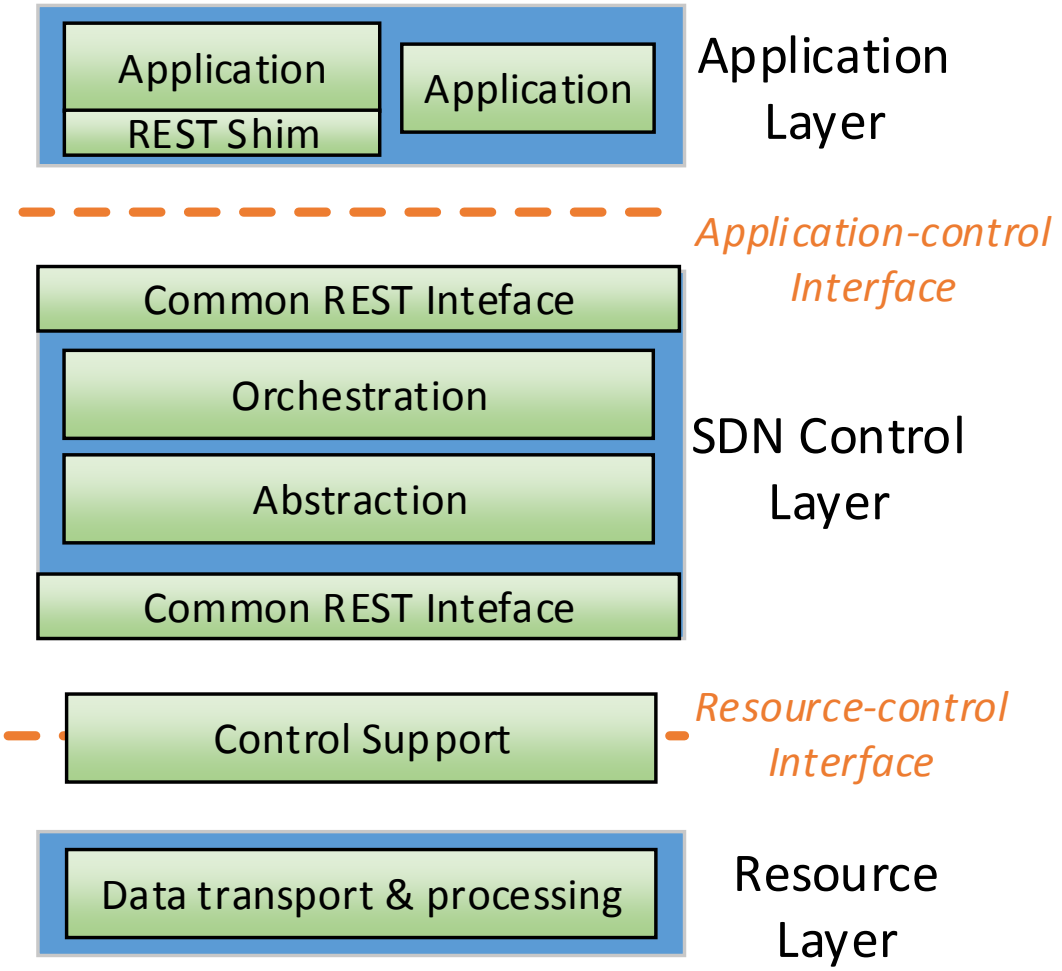

Figure 2 - Proposed SDN Transition Architecture

Note the modifications and additions:

1. The "Application Support" layer is removed in favor of a common standards-based NETCONF/RESTCONF interface, for the core used on both the NBI and SBI.
2. Applications that require it will have a REST "Shim" provided to convert their native requests into the common REST language of the new core.
3. The SDN Control Layer (the core) will perform request routing and minimal modifications to the incoming requests in early implementations.
4. The SBI will interface to the "Control Support" portion of the resource layer, which will utilize ODL as the interface management system between the core and the network resources. Because it is "independent" of the network resources (i.e. not a resource itself or an EMS), it will act as the interface between the core and the network but independent of both.

5. Advantages

Common Interface Language. By using REST as a common interface language for the application layer, any new applications will have a well-defined and extensible interface to the core and can access the core directly via the "application support" layer. By adhering closely to existing YANG and NETCONF standards, the interface will be well documented and accessible.

Limited State in the Core. Since Applications are currently managing their own state anyway, stateful information in this architecture is effectively pushed to the edges: Applications will maintain their state, and the edge interface to the network (e.g. in this case, ODL) will maintain state associated with requests made to network resources. This minimizes HA/DR requirements on the core focusing instead on load balancing and total throughput capabilities.

Inherent Modularity. With state managed at the edges, the architecture allows for a high degree of modularity. While a monolithic approach is still possible, modularity is the order of the day:

- The core's primary function is routing requests to the correct network interface (i.e. ODL). As such, pan-network knowledge is not required and multiple independent instances can be spun up to handle incoming requests.
- Incoming request load balancing and HA/DR is straightforward: Send the requests to any available server. If something is down, spin up a new one.
- ODL functions as an interface between the core and the network resources, but can be limited to a subset of network devices (1-to-1 ODL to resource possible though unlikely). By capping the number of devices managed by a single instance, state management and replication (for HA/DR) can be set to a realistic size based on the maturity/capabilities of ODL.

Feature Extensibility. By separating the request routing from resource interface management and creating modules for each, feature upgrades and enhancements become a piece-wise exercise. Upgrades can be done one section at a time, limiting exposure in production to a well-defined set of devices and simplifying rollbacks as well.

6. Example Architecture

Figure 3 shows a simplified example of how the architecture might look in production.

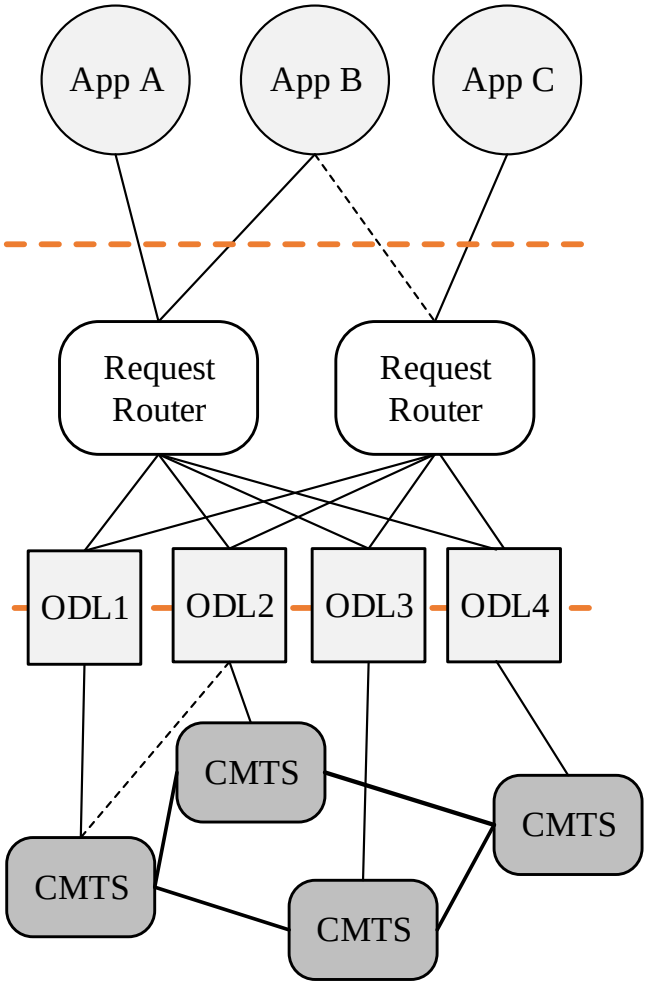

Figure 3 - Example Implementation

At the top, Applications A, B and C make requests into the core. For PCMM, Applications include Diameter-based telephony requests, Fairshare, etc. Application B is shown with a backup link to a second request router. The Applications use the common REST interface.

In the core there are 2 “Request Routers,” more are possible as they work relatively autonomously. These decide which ODL instance to forward the request to. All “routing” here is done on REST requests.

As the interface to the network ODL receives the requests and manages the interaction with the network, each ODL would be associated with a subset of the CMTSs. ODL2 shows a backup connection to an additional CMTS. In the event that the primary ODL for this CMTS is down, requests could be rerouted to this ODL if extra capacity is provided to handle the additional load.

7. Evolution

Currently, the core is planned to be more or less a request router, relying on the applications to provide more detailed “control” of transactions. While the core will ultimately manage the implementation of a

given transaction, the higher-level management will be left to the calling application. In the future, the core will need to possibly do the following:

- Enhanced, condition-based routing modifications
- Modification of request contents based on observations/rules (e.g. Time of day/event-based mods)
- Automatic management activities (not based on requests, self-initiated)
- Injection of these changes in real time.

SDN has its roots as a routing management platform that can manage network routers and provide super-set rules for routing and control. The proposed architecture supports this same context in the core: Create a super-router with rules/functions that can manage and provide rules to other core request "routers," and thus provide additional features and functions. The core becomes a mini-SDN environment, with the request routers as resources and the super-router as the SDN controller for the request "network." This is forward thinking and we will continue to explore and apply what we are currently learning to future designs.

SDN and PCMM

8. ODL PCMM Plugin

The PCMM plugin for ODL, sponsored by CableLabs, was originally conceived to use the ODL platform for the Application Manager and parts of the Policy Server, both components of the PCMM architecture. Specifically, the plugin provides the following features [7]:

- RESTCONF APIs for provisioning CMTS network elements
- RESTCONF APIs for provisioning Service Flow values and types
- RESTCONF APIs for provisioning QoS (or metering) parameters
- PCMM/COPS protocol transport plugin

The initial version of the plugin while functional has some limitations. For instance, it only supports a service class name type gate set, there is limited feedback from the set operation to the original calling application and some extensions to the Yang model will be required to cover all possible gate sets required by a production system. However, the plugin is functional enough with adequate COPS-PR coverage for proof of concept implementations.

9. Use Cases for Programmable Service Flows

Within the cable access network, SDN has the potential to reduce service setup times, reduce errors associated with those setups, offer new management options, and facilitate the introduction of more fine-grained service offerings. While there is a wide array of use cases that SDN can apply to in the cable access network (see [8]), this work focuses on 3 specific use cases:

- Telephony – Specifically the setup and management of QoS associated with calls
- Congestion management – Dynamic, rule-based implementation of bandwidth controls/limits in response to congestion or individual user abuse
- Enhanced video services – Establishing QoS for video services

Theory of Operation

An example request sequence will now be presented.

Referring to Figure 4, Applications at the top corresponding to the 3 use cases from the previous section (Telephony, Congestion Management and Video) make requests into the platform.

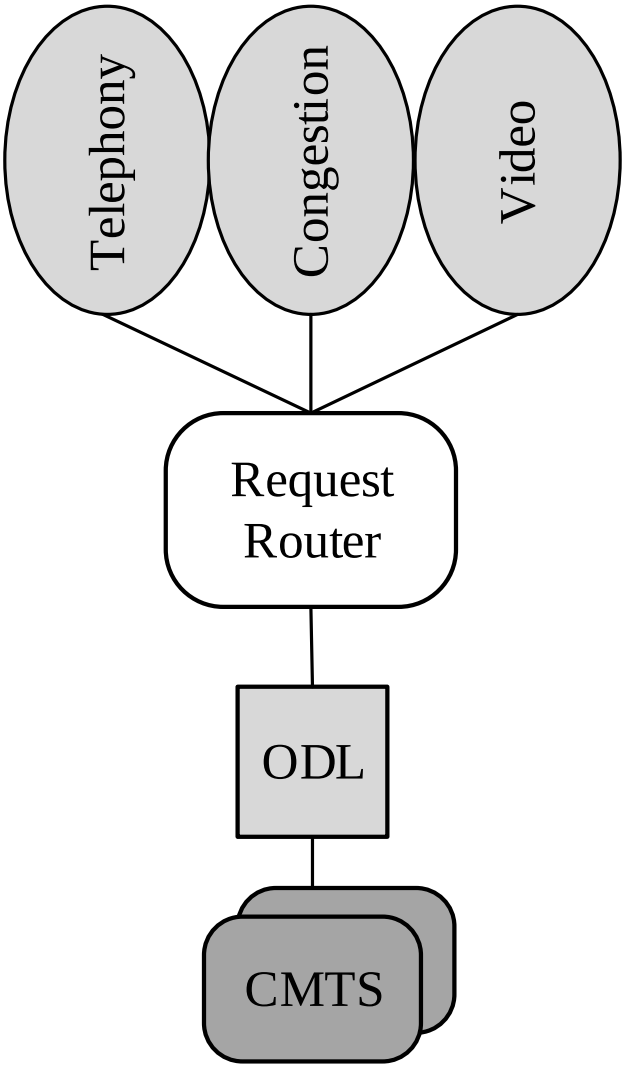

Figure 4 - Request Flow Diagram

The requests are received by the Request Router and transformed (if necessary) into a RESTCONF format for internal handling. Telephony, for example, uses Diameter signaling. The Diameter request is transformed into RESTCONF for forwarding.

The Request Router reads enough of the request to understand to which ODL instance the request needs to be sent. This is basic topological information based on mapping CM IP address spaces to a list of CMTSs.

The request is forwarded to the correct ODL instance, which manages the relationship with the CMTS via COPS-PR. The request will be forwarded to the CMTS with configuration and operational information stored in the local ODL datastores.

10. RESTCONF Request Format

For consistency, the internal RESTCONF format conforms to the existing YANG model-based PacketCable Plugin REST API. An example of the RESTCONF request is provided below. Service requests consist of 2 parts, the HTTP call and the associated JSON data. Simple requests for information are generally just HTTP calls.

All of the information in the HTTP call, and any accompanying JSON data, is supplied by the APPLICATION and not augmented in the current implementation by the core software.

The HTTP call format for a flowspec gate set is as follows:

```
/restconf/config/packetcable:qos/apps/app/<appId>/subscribers/subscrib
er/<CM_IP>/gates/gate/<gateId>/
```

- <appId> - This is a unique label for the requesting agent. This is used internal to the request router core and ODL to identify the origin of a request.
- <CM_IP> - The subscriber IP for the gate request. This address is used to locate the correct CMTS and thus the correct ODL instance.
- <gateId> - This is an internal gate identifier used in ODL to uniquely identify a gate. Note that this is a highly overloaded term and is NOT the gateId returned by the CMTS. The gate identifier returned form the CMTS is stored in ODL but is NOT used for internal identification.

Figure 5 shows example JSON content for the flowspec gate set. This type of gate required an extension of the existing packetcable plugin, which out of the box only supported a service class name gate definition. The information is self-explanatory, with the one note that the <gateId> MUST match the gateId used in the HTTP call.

```
{
    "gate": {
        "gateId": "<gateId>",
        "classifiers": {
            "classifier-container": [
                {
                    "classifier-id": "1",
                    "classifier": {
                        "srcIp": "10.20.0.3",
                        "dstIp": "10.20.0.5",
                        "protocol": "0",
                        "srcPort": "54322",
                        "dstPort": "4322",
                        "tos-byte": "0x01",
                        "tos-mask": "0x00"
```



```
                    }
                }
            ]
        },
        "gate-spec": {
            "dscp-tos-overwrite": "0x00",
            "dscp-tos-mask": "0x00",
            "direction": "us"
        },
        "traffic-profile": {
            "flow-spec-profile": {
                "token-bucket-rate": "400",
                "token-bucket-size": "40",
                "peak-data-rate": "400000",
                "minimum-policed-unit": "400",
                "maximum-packet-size": "400",
                "rate": "30000",
                "slack-term": "0"
            }
        }
    }
}
```

Figure 5 - Example JSON Content for a Flowspec Gate Set Request

11. ODL Request Sequence

There are several steps that a request goes through internal to ODL, including the creation and management in the internal configuration and operational datastores as defined in the NETCONF RFC [9].

When a request arrives at an ODL instance, the following internal communications occur (See Figure 6):

1. Basic verification of the content of the request is made against the relevant YANG models. If a mandatory element is missing, the request is rejected and an error returned to the caller.
2. Upon acceptance of the request, an entry is made in the Configuration datastore and a “200 OK” message is returned to the sender (no additional information is provided to the sender).
3. Using the COPS-PR protocol, the request is forwarded to the CMTS.
4. The CMTS either accepts and establishes the gate or rejects it. Upon acceptance, a gate ID is generated by the CMTS that is unique for the gate on that CMTS
5. If a gate ID is returned, the established gate information, including the gate ID, is stored in the operational datastore. If no gate ID is returned, error information is stored in the operational datastore.

12. Management Considerations

As currently implemented, ODL does not return any confirmed operational information to the requesting application. The only message is a “200 OK” message indicating correct syntax and acceptance of the

request. Any additional information about the gate, including whether it was successfully set, must be explicitly requested by the calling application.

Lessons Learned

Open source has its challenges. Open source holds a lot of promise for rapid development and relatively cheap software options. However, there are limitations. Acceptable operational practices in one industry may not translate to others. And not all development decisions are good. One of the challenges we ran up against was a "native" need for OpenDaylight to access the internet on first start up due to dependency issues. This can be resolved, but it required work on our part to create an environment that did not require this functionality.

Open source needs time to mature. The PCMM plugin functioned, but was far from complete. Once again, just because software has the right label and uses the right terminology doesn't mean it is ready for prime time. Be prepared for additional work to bring the parts you need up to speed.

Just because it says "Java" doesn't mean it is good. Java has many good things going for it, but just because it is uses Java doesn't mean it is well-written. Because it is somewhat ubiquitous, Java suffers a bit from a surfeit of average to below average programmers. Be aware that anything "industry standard" doesn't mean "industry leading."

Care needs to be taken not to overload terminology. The code may not care but the ease of management is reduced if operators have to work to understand and debug requests or logs because of ambiguous terminology. "GateID" turned out to be particularly overloaded in OpenDaylight and the PCMM plugin. Pick terms and stick to them and make sure that you leave no ambiguity for your organization.

Conclusion

SDN is a necessary and inevitable eventuality for the evolution of broadband networks. The agility gained and potential for market differentiation will drive this change.

Moving SDN into DOCSIS driven networks will require some adaptation of traditional network tools to the special requirements of the broadband provider but current experience shows that this is very possible and realistic and measureable results are possible. With the application of practical knowledge and some industry common sense, MSOs can leverage some tools without starting from scratch and start to analyze and understand the potential of SDN in their networks.

Abbreviations

AP	Access Point
bps	Bits per Second
CM	Cable Modem
CMTS	Cable Modem Termination System
CCAP	Converged Cable Access Platform
DPD	Downstream Profile Descriptor
FEC	Forward Error Correction

HFC	Hybrid Fiber-Coax
HD	High Definition
Hz	Hertz
SCTE	Society of Cable Telecommunications Engineers
ISBE	International Society of Broadband Experts
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiplexing with Multiple Access
PNM	Proactive network maintenance
SDN	Software Defined Networking
SD	Software Defined
SF	Service Flow

Bibliography & References

[1] N. Feamster, J. Rexford, E. Zegura. The Road to SDN: An Intellectual History of Programmable Networks. IEEE ACM Queue, December 2013.

[2] DOCSIS 3.1 MAC and Upper Layer Protocols Interface Specification, CM-SP-MULPIv3.1-I02-140320, CableLabs, 2014

[3] DOCSIS 3.1 PHY Physical Layer Specification, CM-SP-PHYv3.1-I07-150910, CableLabs

[4] OpenFlow® [Online] Available from: http://openflow.com

[5] J. Tourrilhes, P. Sharma, S. Banerjee and J. Pettit, "The Evolution of SDN and OpenFlow: A Standards Perspective," ONF, 2014

[6] Recommendation ITU-T Y.3300 (06/2014) Framework of software-defined networking, ITU, 2014

[7] OpenDaylight Users Guide, https://www.opendaylight.org/sites/opendaylight/files/bk-user-guide.pdf

[8] SDN Architecture for Cable Access Networks Technical Report, CableLabs, VNE-TR-SDN-ARCH-V01

[9] RFC6241: Network Configuration Protocol (NETCONF), IETF

