SCTE·ISBE CABLE-TEC EXPO 2017
DENVER, CO
OCTOBER 17-20

2017 Fall Technical Forum
SCTE·ISBE • NCTA • CABLELABS

A Practical Approach to Virtualizing DOCSIS 3.1 Network Functions

A Technical Paper prepared for SCTE/ISBE by:

David S. Early
Data Scientist
Applied Broadband, Inc.
2741 Mapleton Ave
Boulder, CO 80304
720-470-7460
david@appliedbroadband.com

Paul E. Schauer
Distinguished Engineer
Comcast
183 Inverness Dr W,
Englewood, CO 80112
303-372-1215
paul_schauer@comcast.com

Jason K. Schnitzer
Founder
Applied Broadband, Inc.
2741 Mapleton Ave
Boulder, CO 80304
720-838-4465
jason@appliedbroadband.com

Table of Contents

List of Figures

List of Tables

Introduction

There is broad consensus amongst the networking community that programmable network architectures (including SDN and NFV) represent the next stage of connected infrastructure evolution. Benefits for providers are many, stemming from innovative new approaches that re-factor the development and deployment of networks and services.

The service provider's ability to rapidly and continuously develop and deploy new software and tools necessary to realize these objectives is central to the success of this model. Not only will the network see technical change with the introduction of virtualization, but the service provider's organization will require significant changes as well.

This paper presents a structured approach to the evolution of virtualization within the service provider's practice. We establish criteria for the virtualization of network functions within the broadband access network. In doing so, we provide system considerations for the tradespace between the use of centralized and distributed deployment architectures. We provide an overview of Service Provider DevOps (SP-DevOps) and how it can be applied within the cable service provider environment for the continuous deployment of virtualized networks and services.

We present a practical example to illustrate key concepts, developing a simple microservice that provides an implementation of the CableLabs DOCSIS 3.1 Common Collection Framework (DCCF) software system. Use of this this DOCSIS 3.1 microservice will be shown using a container architecture within a cable operator provider's cloud infrastructure.

Service Provider DevOps

In contemporary software development, it's almost impossible to ignore the term DevOps, a portmanteau of "Development" and "Operations." DevOps represents a practice within Information Technology (IT) that defines an organizational model for collaboration between software development and software operations.

Though DevOps is a popular topic in software engineering research, it currently lacks a widely accepted standard definition. The following is a common definition of DevOps based on a novel analysis of peer-reviewed articles:

"DevOps is a development methodology aimed at bridging the gap between Development and Operations, emphasizing communication and collaboration, continuous integration, quality assurance, and delivery, with automated deployment utilizing a set of development practices". [1]

The goal of DevOps is to "bridge the gap" between the internal functions responsible for producing software and the functions charged with running it. Major Internet software, services, and cloud provider companies have adopted DevOps practices to improve the velocity and quality of software development and delivery within their practice [2][3].

With success as a model for software development in large internet enterprises, it has been proposed that DevOps may be useful within the service provider environment as well (including DOCSIS access networks). Though many of the DevOps concepts readily apply, it is worth examining some key differences between target infrastructures. Broadband access service providers have the following distinct

requirements, relative to most other organizations implementing DevOps principles. Notable differences between service provider access network environments and Internet enterprise networks include:

- *A high cost of operation in terms of time and human/financial resources due to the physical management of distributed network nodes. Access Network service providers supervise a management domain that extends beyond a centralized facility (e.g. Data Center), to intelligent devices at the edge of the network with execution environments extending through the cable plants and into each broadband subscriber's home.*

- *Limited visibility of distributed network and service states that make it difficult to assure the Quality of Experience (QoE).*

- *Difficulties in pinpointing the cause and location of problems (troubleshooting) and debugging.*

- *Difficulties in deploying services quickly and frequently, e.g. due to the validation of service integration or regression testing.*

This short list of maladies, shared within large scale Information Systems (ISs) development and operations, is not unlike those faced by service providers when capturing the limitations born of current network management systems and organizational practices. In recognizing this, service providers and network engineering researchers have begun to evaluate the applicability of DevOps practices to analogous challenges of scale and complexity inherent in both traditional (hardware physical element centric) and more contemporary (software virtual element centric network architectures.

It has been noted that though similar management automation goals exist within the programmable network infrastructure domain, service providers face challenges unique to access network systems. Where DevOps was formed from IT organizations working within concentrated data center environments, large service providers operate an inherently more geographically distributed system. Project UNIFY [4] proposes a list of four key characteristics of telecommunications networks making them different from IT organization data centers. These include:

- *Higher spatial distribution with lower levels of path and equipment redundancy;*
- *High availability;*
- *Strictly controlled latency;*
- *Larger number of distributed datacenters.*

However different, greater similarities have motivated new research into the applicability of DevOps principles in a service provider environment. When applying key concepts of DevOps to the Service Provider domain, the integration of these models took the form of Service Provider DevOps (SP-DevOps). The adjusted model proposes the following aspirations:

- *Iterative Development / Incremental feature content*
- *Continuous deployment*
- *Automated processes*
- *Holistic/Systemic views of development and deployment/operation.*

The UNIFY project represents a first attempt to codify SP-DevOps practices in the form of a standards-based approach. The initiative aims at applying DevOps concepts to telecom operator networks and supporting the idea of fast network reconfiguration.

Microservices:

Building Modular Distributed Applications

Microservices are small, purposeful modules of software executing in a distributed environment. Microservices can be used to build larger applications (northbound) that implement specific business solutions through interactions with the network operator's back-office infrastructure. The microservices architectural model is heralded to be a more agile approach to complex system development while being better aligned with the goals of SP-DevOps and a continuous deployment model.

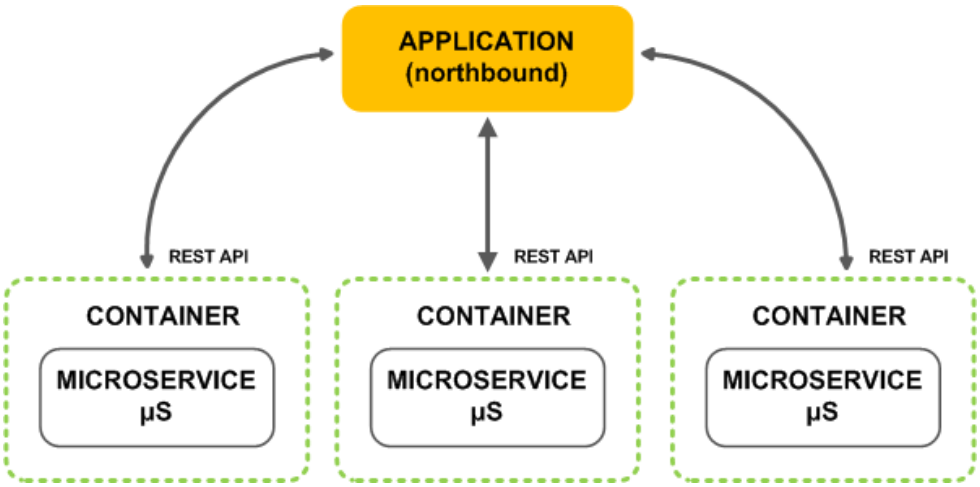

Figure 1 - Building Networked Applications from Microservices and Containers

Containers:

Practical Architecture for Network Virtualization

Containerization, also called container-based virtualization, is an OS-level virtualization method for deploying and running distributed applications without launching an entire Virtual Machine (VM) for each application. Instead, multiple isolated systems, called containers, are run on a single control host and access a single kernel.

Microservices are often executed in containers. Containers offer a new form of system virtualization that significantly reduces the resource cost and complexities of their VM predecessors. Where VMs embody a complete executing environment and copy of the OS, containers perform execution isolation (virtualization)

at the OS level. In this way, a single OS instance can support multiple containers, each with a microservice executing inside [5].

The Rise of REST

The REpresentational State Transfer (REST) protocol has risen to become the de facto choice for web development with the evolution of Inter-Process Communication (IPC) to make use of text-based serialization formats, like XML and JSON. Protocols such as SOAP allowed IPC across HTTP, and soon web developers were not just building web applications that served content to browsers, but web services that performed actions and delivered data to other programs. This services-based architecture proved to be very powerful, as it eliminated dependencies on shared code libraries, and allowed application developers to further decouple their application components. The SOAP protocol and the related WS-* standards soon became increasingly complex and heavily dependent on specific implementations in application servers, so developers migrated to the much lighter-weight REST protocol. As the use of mobile devices exploded, and as web interface development switched to AJAX and JavaScript frameworks, application developers started to make extensive use of REST for transmitting data between the client devices and the web servers.

In popular Software Defined Network (SDN) architectures such as OpenDaylight [6] and ONOS [7], REST has become a critical infrastructure component, extended by the IETF to support configuration management concepts by offering a lighter-weight transport to NETConf's more cumbersome transactional model [8]. As we will see in the example provided, the use of REST lends itself well to microservices based architectures and to key SP-DevOps development and delivery principles.

DOCSIS 3.1 Management Data

1. Proactive Network Management (PNM)

The DOCSIS 3.1 Physical Layer (PHY) specification introduces a new network management technique for gathering highly valuable operational data from CCAP and CM devices [9]. The Proactive Network Maintenance (PNM) data can be sourced from the CCAP, the CM or as a collaboration of both CCAP and CM and for larger data sets delivers bulk data faster than traditional data collection methods (e.g. SNMP) via TFTP file transfers. **Error! Reference source not found.**2 illustrates the various test points and test functions defined in the specification (sourced from [9]).

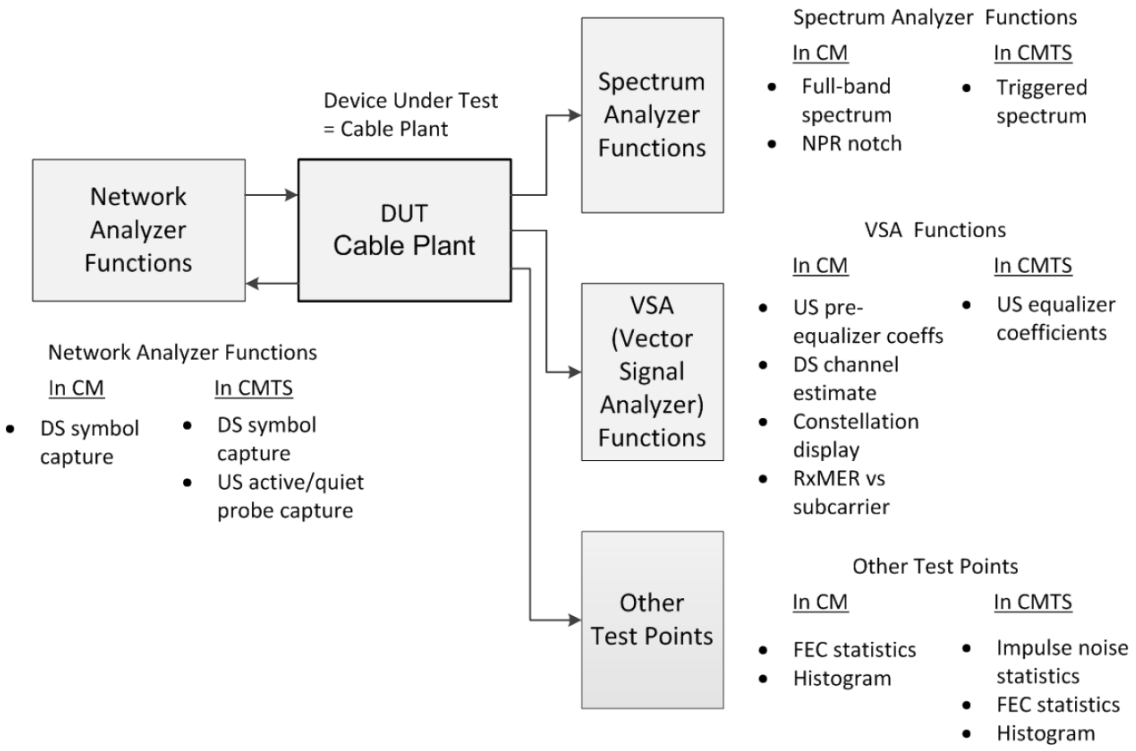

Figure 2 - CM and CCAP test points as illustrated in [1](Figure 9.1)

These features provide capabilities similar to test and measurement equipment without the limitations associated with this type of testing (e.g. cost, limited deployment, limited access). By placing the test functions in the system to be tested and providing easily accessible methods of data acquisition, PNM data represents a valuable source of operational information for the MSO to identify physical layer anomalies and proactively address issues that affect access network performance and service quality.

1.1. DOCSIS 3.1 PNM Data Collection Workflow

Collection of DOCSIS 3.1 PNM data from a DCOSIS 3.1 CM device involves the following steps:

1. SET via SNMP any configuration parameters for the test:
    a. Configuration parameters may include offsets, windows, timeouts, etc.
    b. For tests that return a file, set a filename and TFTP service IP.

2. Trigger the test via SNMP set to the CM device:
    a. All tests include a “TriggerEnable” field to initiate the test.

3. Collect the data:
    a. For file-based data, part of the SET step is setting a TFTP destination. The data, when complete, will be forwarded to this TFTP service
    b. For SNMP based data, the data must be retrieved from the device via SNMP.

These steps are non-trivial, order-dependent steps for obtaining PNM data – making this a prime candidate for a distributable microservice providing a uniform user interface for gathering and using PNM data.

1.2. D3.1 CM PNM File Data

Note that the specific definitions of these data sources can be found in [10] and [11]. Please refer to the most recent version of this document for a full definition of the test, its data sources, and content.

Table 1 - Defined CM PNM Tests

PNM Test	Source	Note[1]
CM Symbol Capture	PNM File	Analyze the response of the cable plant from the CM's perspective based on a sample symbol captured at the CCAP and CM. Paired with the CCAP equivalent below.
CM Channel Coefficient Estimates	PNM File	CM estimate of the downstream channel response coefficients, typically used for the CM's downstream equalizer.
CM Ds Constellation Display	PNM File	CM downstream constellation display providing received QAM constellation points.
CM Ds OFDM Rx MER	PNM File	Measurements of the receive modulation error ratio (Rx MER) for each subcarrier
CM Ds Histogram	PNM File	Measurement of nonlinear effects in the channel such as amplifier compression and laser clipping. CM captures the histogram of time domain samples at the wideband front end of the receiver (full downstream band).
CM Pre-equalizer Coefficients	PNM File	CM upstream pre-equalizer coefficients. The CM pre-equalizer coefficients and the CMTS upstream adaptive equalizer coefficient update values, when taken together describe the linear response of the upstream cable plant for a given CM.
CM FEC Summary	PNM File	A series of codeword error rate measurements on a per profile basis over a set period of time (10min or 24hr).
CM Spectrum Analysis	PNM File	CM downstream spectrum analysis function, each measurement is a data collection event that provides the energy content of the signal at each frequency within a specified range.
CM OFDM MER Margin	SNMP	An estimate of the MER margin available on the downstream data channel with respect to a candidate modulation profile. This is similar to the MER Margin reported in the OPT-RSP Message [MULPIv3.1].
CM OFDM Required QAM MER	SNMP	Calculated Required Average MER based on the bit loading for the profile and the Required MER per Modulation Order provided in the CmDsOfdmRequiredQamMer Table.

[1] Paraphrased from [7]

2. DOCSIS Common Collection Framework (DCCF)

Introduced early in 2017, the CableLabs DOCSIS 3.1 Common Collection Framework (DCCF) was introduced to abstract the complexity of low-level data collection in DOCSIS 3.1 network deployments.

CableLabs Proactive Network Maintenance (PNM) [PNM] program has been met with measurable success by Cable operators working to enhance best practices for DOCSIS 3.0 network operations. With the introduction of D3.1, it is anticipated that PNM adoption will increase while network data complexity and volumes will rise.

In addition, new platforms that enhance visibility into other critical segments within the service provider access network infrastructure, both wireless and wireline, will also be addressed by companion PNM initiatives. In doing so, the Cable operator will be enabled with a common platform and methodology upon which to build enhanced applications to support current and future operational use cases. Potential network infrastructures within the operator's management domain that would benefit from common PNM practices could include Wi-Fi, MOCA, R-PHY, and optical.

It is expected that multiple network technologies will be under PNM management within an operator's infrastructure at the same time. It is also understood that different network types will expose different forms of operational instrumentation based on information models inherent to their design and deployment disposition. In addition, over the course of broadband network evolution, a number of different network management protocols have been adopted to manage D3.1 networks.

Though data is collected from the same network, it is often gathered by multiple protocols (SNMP, TFTP, SYSLOG) embedded in disparate and closed network management systems. This adds to the burden of network data collection, making holistic visibility unattainable.

In order to motivate wider adoption of PNM practices across current and emerging network technologies, it has been proposed that a structured approach to network data collection would accelerate development and deployment by abstracting the complexities of multi-network visibility through the support of standard network information models and protocols. The Common Collection Framework (CCF) provides a structured approach to the collection of data from standards-based network deployments.

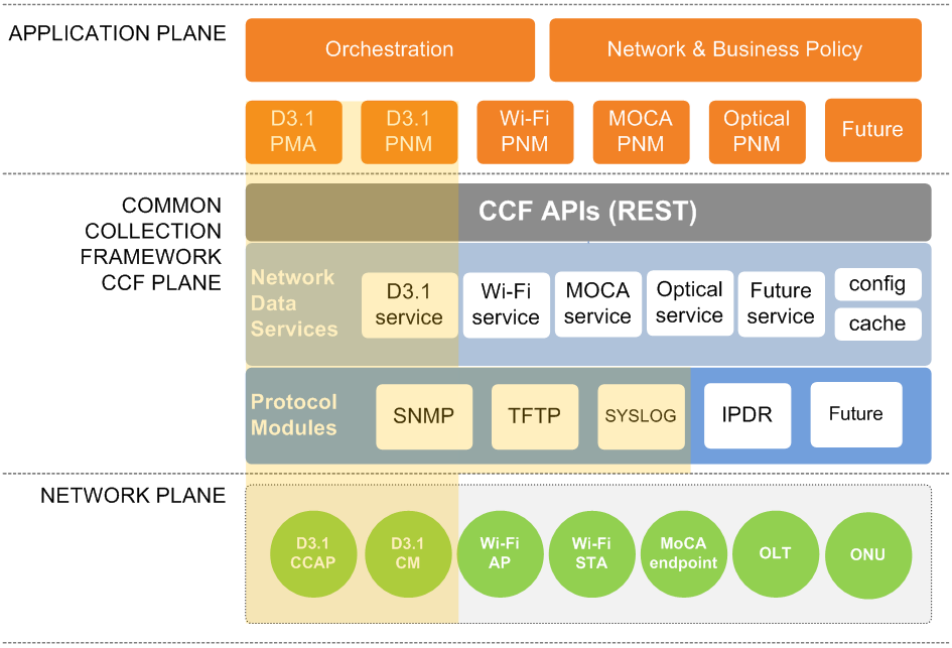

Figure 3 - CableLabs Common Collection Framework Architecture

As the name indicates, the DCCF is concerned with connecting D3.1 management applications with data instrumented by the underlying D3.1 network plane. This restricts the DCCF to the following subset of logical components described in the CCF architecture:

- D3.1 PMA and D3.1 PNM applications.
- D3.1 network data service.
- SNMP, TFTP, and SYSLOG protocol modules.
- D3.1 CCAP and CM devices.

The overall DCCF architecture follows the model described for CCF in the preceding sections.

An Example DOCSIS 3.1 Virtualized Microservice

In this section, we bring together the concepts introduced earlier in the form of a basic DOCSIS 3.1 virtualized microservice using the DCCF software system.

3. DCCF Microservices Architecture

3.1. Software Modules

The DCCF design is well-suited for deployment as a scalable microservice. DCCF consists of three (3) primary modules:

1. **DCCF REST API (RA) -** All user REST requests are directed to the DCCF RA. The RA acts as a request router, forwarding any incoming requests to the correct destination. Requests are forwarded to the Workflow Controller via an internal REST interface.

2. **DCCF Workflow Controller (WC) -** The WC manages the execution of individual tasks against network devices or external data sources.
    a. The WC contains "Drivers" for different network operations. The first driver is an SNMP driver. Future drivers might include:
        i. Interfaces to provisioning systems
        ii. Data retrieval from other data sources (DB, other collection systems)
        iii. IPDR collection (most likely through file import)
    b. Each driver is made up of one or more driver modules that performs an action. These are interchangeable as long as the input and output formats remain the same. This means that in most cases, a single action/function can be updated without resetting the system.

3. **DCCF TFTP Service (TFTP) -** This is a TFTP service customized for specific file management functions required by DCCF. It is a service for requests to PUT files from remote devices such as CMs, and for GET requests from the DCCF for retrieving remove file.

4. **DCCF Disk Cache (DC) -** This is the local data storage associated with each WC. The RA does not have a DC. The DC is not meant to be long term storage, and any long-term storage needs to be done external to the DCCF. The DC has no software component.

Each module of the DCCF can be updated independent of the others, as long as the interface characteristics (parameters in and out) remain the same.

4. DCCF Deployment and Scale

Error! Reference source not found.4 shows a simple standalone DCCF installation, with all modules located in a single container. An obvious deployment case for testing, it also represents the most atomic deployment possible with an assumed 1:1 relationship with a CCAP. Using an external proxy router for incoming requests, an entire production environment could be made of small, more atomic installations.

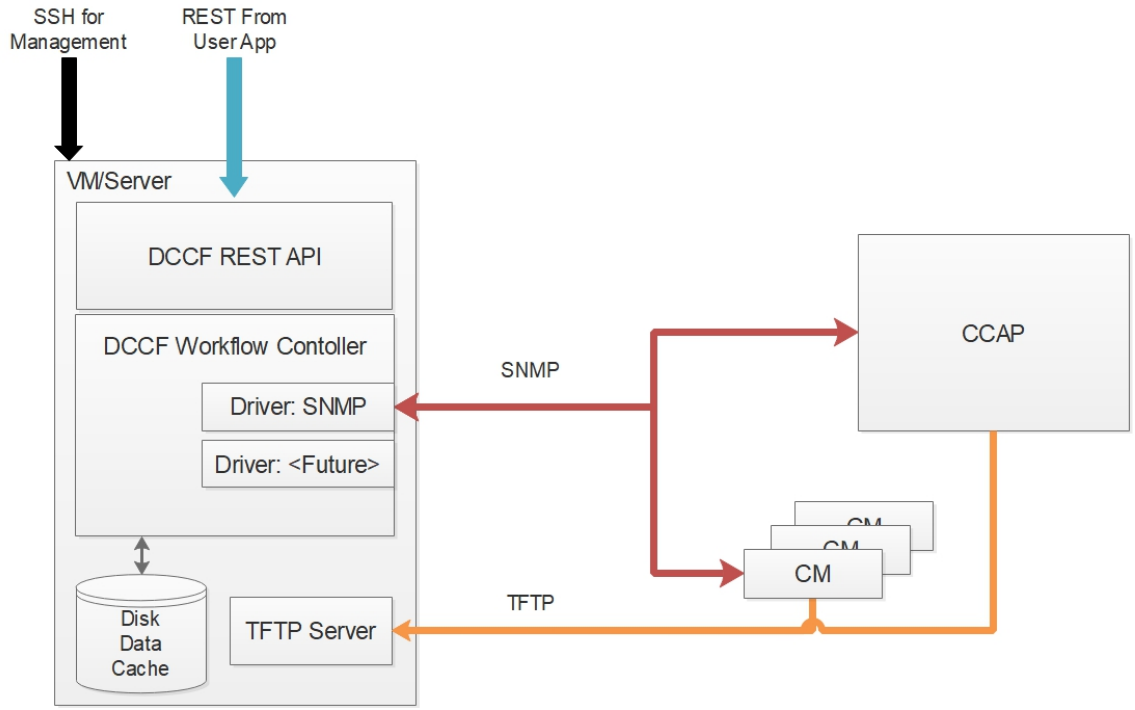

Figure 4 - Basic DCCF installation

As each module is a standalone entity, they may be distributed in different containers. **Error! Reference source not found.** shows a distribution of the RESTful, WC, and TFTP services.

This configuration introduces the concept of a remote TFTP service which requires a new Driver module, a "GetTFTP" module, to retrieve files from the remote TFTP service. When a file arrives from a remote device, the TFTP service sends a REST alert to the WC which triggers a GetTFTP action to retrieve the file and store it in the local cache.

Configuring a remote TFTP service requires some minor changes to the DCCF configuration file. Also note this example maintains a 1:1 relationship between RA and WC: One RA serves one WC.

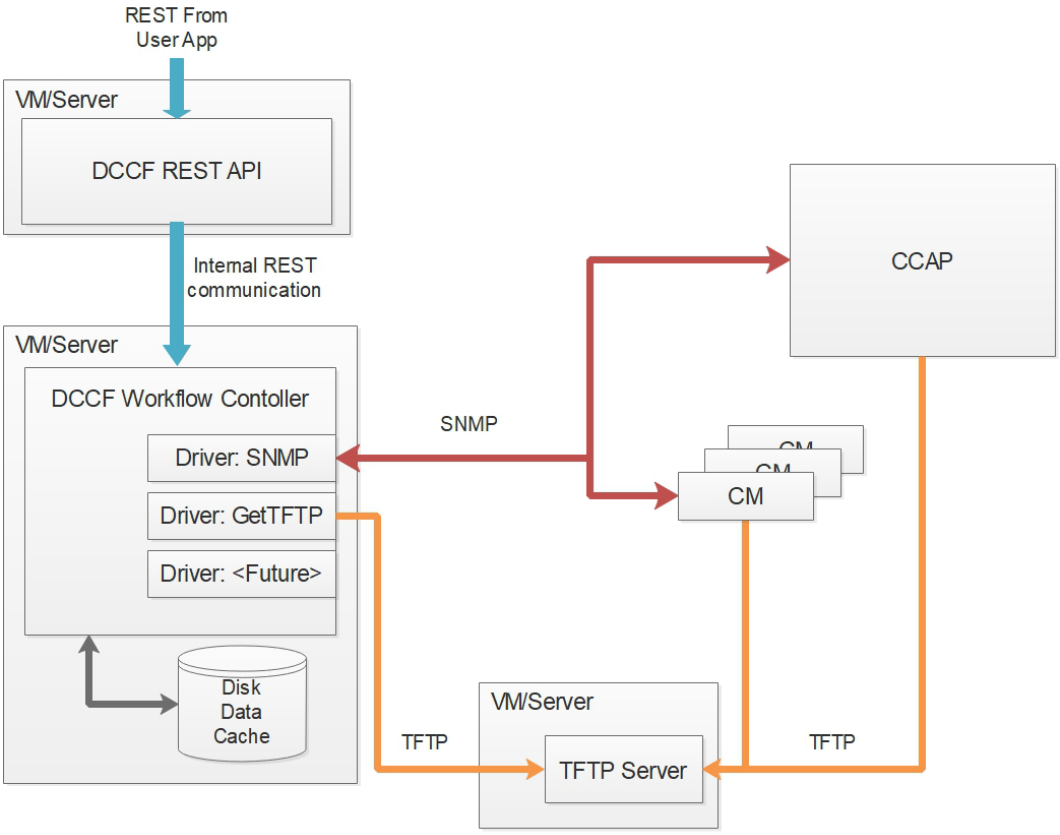

Figure 5 - Distributing the DCCF modules

Finally, a fully distributed deployment is illustrated in **Error! Reference source not found.**. This is not possible with the current version 1.0 of DCCF, due to an immature routing function in the RA, but is totally supported with the current architecture:

1. Multiple RAs, mostly likely served by a simple commercial load balancer, take incoming user REST requests.

2. The routing in the RA distributes the requests so the correct WC.

3. The WC performs the actions requested, storing and making available data in the local DC.

4. TFTP activity is managed through one or more remote TFTPs (each WC configured to use an appropriate TFTP service).

By controlling the resources associated with the virtualized WC, this deployment strategy gives excellent horizontal scaling options as well as offering multiple options for physical distribution in the operations network and container environment.

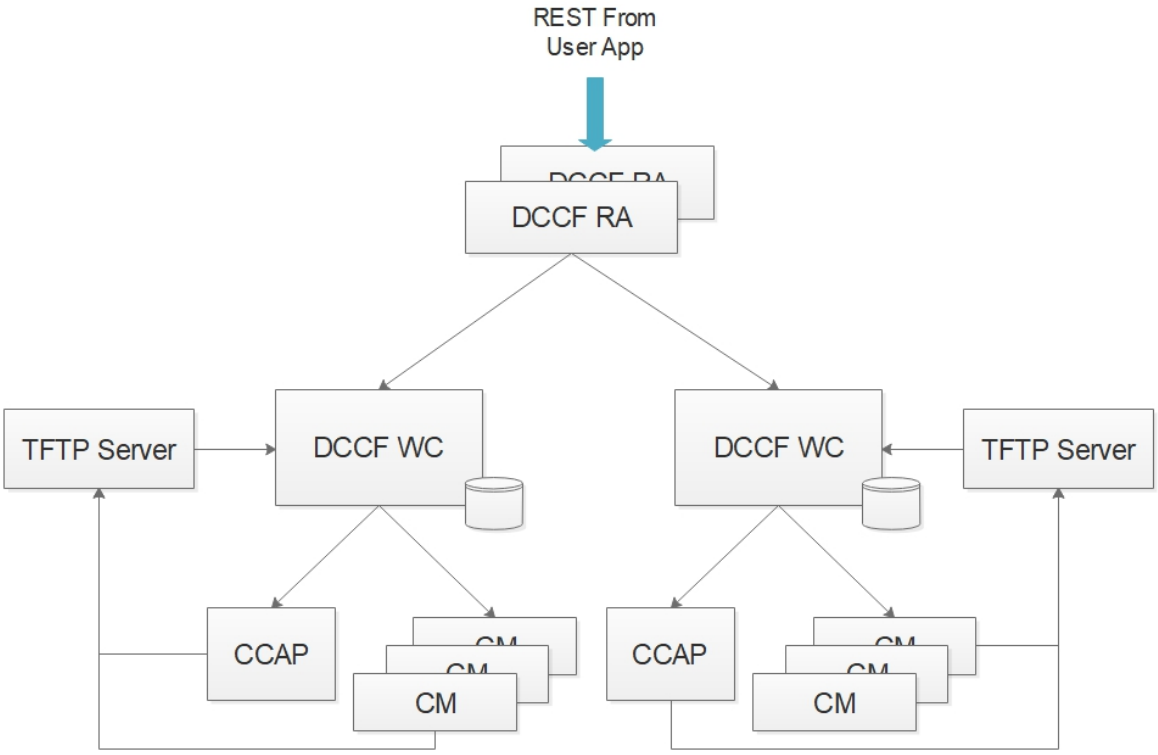

Figure 6 - Distributed DCCF with Multiple TFTP Service containers and Load Balancing

5. DCCF Features and Functions

The DCCF REST API implements a growing number of useful commands for the exploration and management of DOCSIS 3.1 networks. Table 2 shows a summary of available DCCF operations available as of release 1.0.

Table 2 - DCCF v1.0 REST API commands

Operation	Description
POST /dccf/ccaps/(CCAP)/cmPNMDsRxMer	Initiate collection of cmPNMDsRxMer data for CMs devices listed in attached JSON file.
POST /dccf/ccaps/(CCAP)/cmPNMFecSum	Initiate collection of cmPNMFecSum data for CMs listed in attached JSON file.
POST /dccf/ccaps/(CCAP)/initialize	Add new CCAP to DCCF.
POST /dccf/ccaps/(CCAP)/registered31cms	Create/update list of D3.1 CMs on CCAP.
POST /dccf/ccaps/(CCAP)/topology	Create/update CCAP topology information (Multiple GET options).
POST /dccf/ccaps/(CCAP)/cms/(CMMAC)/cmDeltaDsFecStats	Poll the latest cmDeltaDsFecStats (SNMP) data (GET retrieves data).
POST /dccf/ccaps/(CCAP)/cmPNMDsRxMer	Initiate collection of cmPNMDsRxMer data for CMs listed in attached JSON file (GET retrieves data).
POST /dccf/ccaps/(CCAP)/cms/(CMMAC)/cmPNMDsRxMer	Initiate collection of cmPNMDsRxMer data from specified CM (GET retrieves data).
POST /dccf/ccaps/(CCAP)/cmPNMFecSum	Initiate collection of cmPNMFecSum data for CMs listed in attached JSON file (GET retrieves data).
POST /dccf/ccaps/(CCAP)/cms/(CMMAC)/cmPNMFecSum	Initiate collection of cmPNMFecSum data from specified CM (GET retrieves data)
GET /dccf/jobs/(JOBID)	Return status information for specified JOBID.

DCCF Microservice Example

With the concepts of D3.1, PNM, DCCF, and microservices in hand, we can now proceed with a simple example using the DCCF software (release 1.0) running on a virtualized host within a container environment.

The DCCF is running on a host (DCCF_HOST) with SNMP and TFTP access to a DOCSIS 3.1 network. This first query (figure x) is executed in the DCCF client terminal using the **curl** (https://curl.haxx.se) command utility to generate the REST API CM topology discovery command over HTTP. In this example, the DCCF returns a list of all D3.1 CM devices registered on the CCAP (CCAP_IP) is returned describing each by MAC address.

Figure 7 shows a request for a single CM's Downstream Receive MER report. In this example, a client executes a curl command which sends the REST API request to request CM PNM data for the device. The DCCF returns an acknowledgement that the command has been received and is in process.

To retrieve the CM PNM MER measurement data requested, a final REST command is sent via curl from the client's terminal. The command is sent to the DCCF which returns the current data of the CM's Downstream RX MER in an efficiently compressed and archived format.

```
[dccf_host]$ curl -X GET
http://${DCCF_HOST}:8888/dccf/ccaps/${CCAP_IP}/registered31cms
{
  "function": "wc_get_ccaps_registered31cms",
  "json_data": [
    "AC202E772B70",
    "F8A097EF242C",
    "1CABC0B999E4",
    "F8A097EF24B3",
    "64777D90D8C0",
    "64777DE45830",
    "A84E3FCA5B50",
    "1CABC0B99AC6",
    "AC202E772D60",
    "1CABC0B99AF0",
    "1CABC0B99ADC",
    "1056118A0B9E",
    "AC202E7727E0",
    "64777DE45890",
    "64777D5EC500"
  ],
  "message": "wc_get_ccaps_registered31cms: Completed on CCAP Status code: 200",
  "results_in": [
    "json_data"
  ],
  "status": "OK",
  "status_code": 200
}
```

Figure 7 - Retrieving CM Topology Information for a CCAP from DCCF

```
[dccf_host]$ curl -X POST
http://${DCCF_HOST}:8888/dccf/ccaps/${CCAP_IP}/cms/${CM_MAC}/cmPNMDsRxMer
{
  "function": "ra_post_ccaps_cms_pnm",
  "json_data": {
    "function": "wc_route_handler",
    "json_data": "{\"name\": \"cmPnmFile\", \"initialTime\": \"2017-07-20
20:36:20\", \"currentState\": \"ACCEPTED\", \"action\": \"cmPnmFile\", \"jobid\":
\"20170721003620_4b70b0bd_010010010001\", \"ccap\": \"010010010001\",
\"updateTime\": \"2017-07-20 20:36:20\"}",
    "message": "wc_route_handler: Completed cmPnmFile on CCAP",
    "results_in": [
      "json_data"
    ],
    "status": "OK",
    "status_code": 200
  },
  "message": "ra_post_ccaps_cms_pnm: POST
http://${DCCF_HOST}:8888/dccf/ccaps/${CCAP_IP}/cms/1CABC0B99AF0/cmPnmFile/4
workflow_controller response status code: 200",
  "results_in": [
    "json_data"
  ],
  "status": "OK",
  "status_code": 200
}
```

Figure 8 - Requesting CM DS Rx MER Measurement Data From DCCF

```
[dccf_host]$ curl -vv GET
http://${DCCF_HOST}:8888/dccf/ccaps/${CCAP_IP}/cms/${CM_MAC}/cmPNMDsRxMer >
/tmp/pnm_dsrxmer.tar.gz
*   Trying ${DCCF_HOST}...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0*
Connected to ${DCCF_HOST} (${DCCF_HOST}) port 8888 (#0)
> GET /dccf/ccaps/${CCAP_IP}/cms/1CABC0B99AF0/cmPNMDsRxMer HTTP/1.1
> Host: ${DCCF_HOST}:8888
> User-Agent: curl/7.49.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: application/x-tar
< Content-Disposition: attachment; filename=wc_get_ccaps_cms_pnm-cmPNMDsRxMer-
1CABC0B99AF0_MOST_RECENT_None_None.tar.gz
< Last-Modified: Fri, 21 Jul 2017 00:37:33 GMT
< Expires: Fri, 21 Jul 2017 12:37:33 GMT
< Content-Length: 2145
< Date: Fri, 21 Jul 2017 00:37:33 GMT
< ETag: "1500597453.866377-2145-3238667453"
< Cache-Control: max-age=43200, public
< Server: Werkzeug/0.11.11 Python/3.5.2
<
{ [1024 bytes data]
100  2145  100  2145    0     0  81342      0 --:--:-- --:--:-- --:--:-- 85800
* Closing connection 0
```

Figure 9 - Retrieving CM DS MER Measurement Data From DCCF

In this way, we have demonstrated remote interaction with a virtualized DOCSIS 3.1 microservice that provides visibility into the network while abstracting the complexity of low level data collection and topology discovery. DOCSIS network functions and applications can now be developed without requiring low-level access network data collection capabilities. The remainder of this example will present a simple DOCSIS 3.1 application that displays the OFDM DS MER data returned by the DCCF microservice.

To illustrate the CM MER data content, Figure 10 shows visualizations for three CM devices created with a simple Python script that retrieves data from the DCCF REST interface and generates a graph using an open source visualization library.

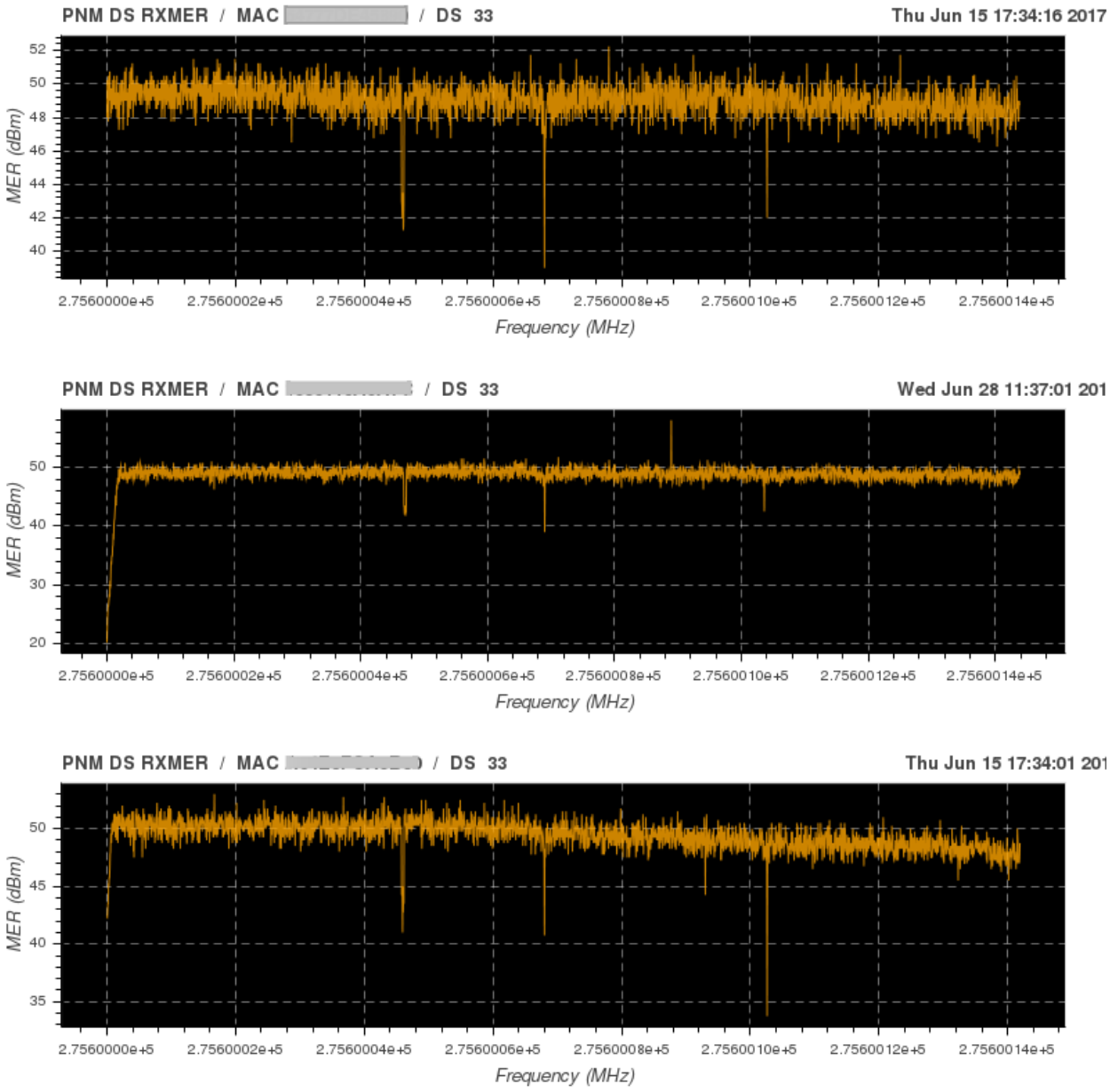

Figure 10 - Example OFDM MER for three different CM devices

Conclusion

We've presented a practical example to illustrate key concepts of virtualizing DOCSIS 3.1 network functions. We presented a simple microservice that provides an implementation of the CableLabs DOCSIS 3.1 Common Collection Framework (DCCF) software system. Use of this DOCSIS 3.1 microservice was shown using a container architecture within a cable operator's cloud infrastructure.

Abbreviations

PNM	Proactive Network Management
CM	Cable Modem
CCAP	Converged Cable Access Platform
DUT	Device Under Test
CMTS	Cable Modem Termination System
SNMP	Simple Network Management Protocol
MSO	Multiple System Operator
DCCF	DOCSIS Common Collection Framework
CCF	Common Collection Framework
MSA	Microservice Architecture
PO	Profile Optimizer
POC	Proof of Concept

Bibliography & References

[1] Teaching Agile Development with DevOps in a Software Engineering and Database Technologies Practicum, 3rd International Conference on Higher Education Advances, HEAd'17 Universitat Politecnica de Valencia, Valencia, 2017, Mason, Robert T., Masters, William and Stark, Alan

[2] From Virtual Machines to Containers and Micro-Services: The Next Generation of Virtualization

[3] http://about.att.com/innovationblog/08252015nextgenerati, August 25, 2015, By Andre Fuetsch

[4] http://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/UNIFY-WP4%20M4_1%20SP-DevOps%20concept%20evolution%20and%20initial%20plans%20for%20prototyping.pdf

[5] https://www.opencontainers.org/

[6] https://www.opendaylight.org/

[7] http://onosproject.org/

[8] https://tools.ietf.org/html/rfc6241

[9] *Physical Layer Specification*, CM-SP-PHYv3.1-I11-170510, May 10, 2017, Cable Television Laboratories, Inc.

[10] *Cable Modem Operations Support System Interface Specification*, CM-SP-CM-OSSIv3.1-I06-151210, May 10, 2017, Cable Television Laboratories, Inc.

[11] *CCAP™ Operations Support System Interface Specification*, CM-SP-CCAP-OSSIv3.1-I09-170510, May 10, 2017, Cable Television Laboratories, Inc.

