Key Learnings from Comcast's Use of Open Source Software in the Access Network

A Technical Paper prepared for SCTE•ISBE by

Louis Donofrio
Sr. Director, Product Management
Comcast Cable
1800 Arch Street, Philadelphia, PA 19103
1-215-283-5023
louis_donofrio@comcast.com

Vignesh Ramamurthy
Principal Architect
Infosys Consulting
1800 Arch Street, Philadelphia, PA 19103
1-510-529-8155
vignesh.vr@gmail.com

Qin Zang
Software Engineer
Comcast Cable
1800 Arch Street, Philadelphia, PA 19103
1-215-283-6878
qin_zang@comcast.com

Table of Contents

List of Figures

1. Introduction

This paper will provide insights and lessons-learned from Comcast's Next Generation Access Network (NGAN) team as we have increasingly deployed open source software (OSS) to virtualize the access network. Using GNU/Linux, Kubernetes, Prometheus, OpenFlow and many other open-source software distributions for automation (Ansible), test (Wireshark) , troubleshooting (smartmon), and Continuous Integration/ Continuous Development (Jenkins, Git), etc., we have achieved some fantastic results. This did not come without a great deal of heavy lifting, new process development, and flexibility. On occasion, it also required re-direction.

By sharing our experience, we hope that other Multiple System Operators (MSOs) will realize the same cost benefits and new feature speed to market we have seen. By providing details on our software selections, others should be able to achieve an even faster pace of innovation as the cable industry moves towards Network Function Virtualization and Software Defined Networking. Key insights we will share include the detailed network visibility gained by using real-time network monitoring tools like Grafana and Kibana, as well as, requirements to use white box switching equipment, commercial off-the-shelf servers and open source operating systems in the network. Last but not least, we will discuss the importance of actively contributing to the open source community while monitoring community developments and building your own internal expertise.

2. Brief History of Open Source Software at Comcast

With the launch of the X1 platform in 2012, Comcast quickly transformed from a multiple system operator dependent on suppliers for innovation to become a software and technology company. Since that time, we have significantly consumed and produced open source software (OSS). Notable Comcast contributions include developing the Reference Design Kit (RDK) platform, which now powers over 60 million consumer premises equipment worldwide. We have also made significant contributions to OpenStack, Apache Traffic Control, and the Apache Hypertext Transfer Protocol (HTTP) Server. OSS users within Comcast include our Video Internet Protocol and Research (VIPER) team who have been using Kubernetes, for cloud DVR and IP Video and have been sharing their insights at conferences including KubeCon and CloudNativeCon since 2016.

In 2016, recognizing the growing importance of OSS, Comcast launched an Open Source Practice Office (OSPO) (figure 1) to drive a more coordinated and planned strategy towards OSS. OSS is a key driver of differentiation, cost reduction and innovation for many leading companies and has become a business imperative for us, as well.

Figure 1 - Comcast's Open Source Program Office

Comcast's OSPO's goals include acceleration of software delivery and focus on core differentiation. Comcast's focus is to leverage open standards for innovation and time-to-market while reducing work on commodity non-differentiating software. We are also using OSS to improve cost efficiency and reduce operating expenses while improving software quality, reliability and security.

3. Open Source Software within NGAN

Comcast's Next Generation Access Networks (NGAN) team uses open source software (OSS) throughout its Distributed Access architecture (DAA). From Ansible to Wireshark, we are leveraging the innovation and speed-to-market for new feature development that comes from participating in community development.

Four key categories of OSS for NGAN include telemetry, containerized applications, automation and operating systems.

Telemetry tools include Grafana, Kibana, Fluentd and Thanos. Containerization tools include Kubernetes to manage Docker containers which house our virtual Cable Modem Termination (vCMTS) application. For automating runbooks, we have deployed Ansible with zero issues. Finally, our first generation DAA utilizes Ubuntu Linux on our servers and the Open Networking Foundation's Open Network Operating System (ONOS) to manage the control and data plane of our Spine/Leaf switching network.

4. Telemetry

Our team realized from the start of the DAA program that the traditional ways of collecting operational statistics regarding the health of our network would not fit the needs of our new architecture. A distributed architecture such as ours was going to be very difficult to troubleshoot and Standard Network Monitoring Protocol's (SNMP's) pull model wouldn't scale for our needs. We wanted to continuously push data to assess the health of the network.

We also wanted to avoid using Command Line Interface (CLI) because they can only be used to check one parameter at a time. With CLI, you need to be very precise with your instructions to get the information needed and there are no graphical views of the information. There were also performance concerns about having too many people logging into critical infrastructure, as well as, security concerns with the likely sharing of passwords.

Using OSS platforms like Prometheus, Grafana, Kibana, and Fluentd for data collection, we were able to provide our operations teams and field personnel with a single pane of glass that provides a view of what is happening across the entire network. Our monitoring probe is recording data every 15 seconds to capture key events and is used to create alarms in Slack to notify support personnel regarding potential issues. Alarms on key parameters are typically triggered after waiting a pre-programmed 5 minutes. As we learn about new types of issues, we continually create alarms to proactively address issues before they become outages.

Any telemetry data being collected can easily be shown in a new Grafana dashboard (figure 2) created as required. Grafana's on-line tutorials make it simple for anyone in NGAN (without programming knowledge or Grafana expertise) to create their own dashboard view.

Internally, we have since settled on seven key dashboards to make sense of the DAA environment and they are typically monitored by hundreds of simultaneous users from divisional business leaders to field technicians. Visibility into network performance and understanding of key network parameters has increased dramatically with these tools.

Figure 2 - Grafana dashboard used for network monitoring

The increased visibility into network performance has been eye opening. We're now aware of every minor glitch in the network. If a technician opens an optical node in the field, we see it. If power goes out in a particular geography, we are aware ahead of any announcement on the power company's website. One particular use case is that of a supervisor viewing Grafana dashboards in the field to ensure that Remote PHY (RPHY) node cuts are going, as planned.

This architecture was not without its issues. There were gaps in telemetry due to bugs and latency. The initial design (see figure 3) had too many components including stream processors. Stream processors were introduced because the virtual cable modem termination system (vCMTS) was not originally sending data in a Prometheus required format.

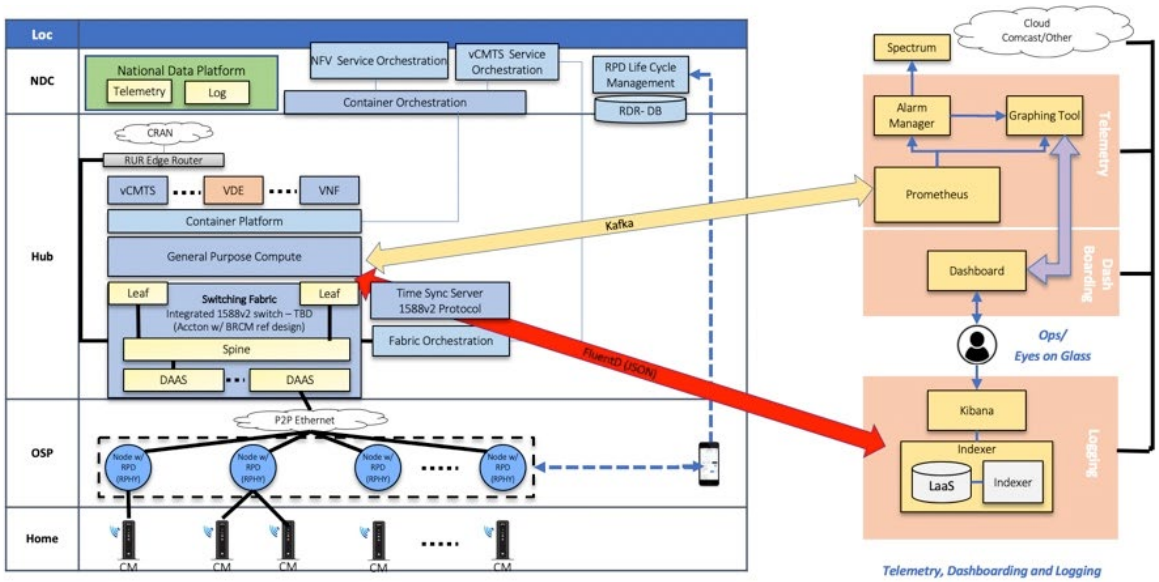

Figure 3 - Original Telemetry Architecture in Comcast DAA

Most of our users are looking at Grafana for telemetry. This simultaneous usage does not slow the vCMTS system as telemetry data is coming from stream processors in AWS but the number of users can affect the load of Prometheus which in turn may affect the response time to Grafana. This is something we are addressing in our next generation telemetry solution.

There are storage costs associated with all these metrics and that has to be considered when architecting the solution. We continue to optimize costs for Amazon Web Services (AWS) including limiting metrics storage to 15 days.

Changes were made to improve scalability as we continually add more R-PHY nodes to the network (figure 4).

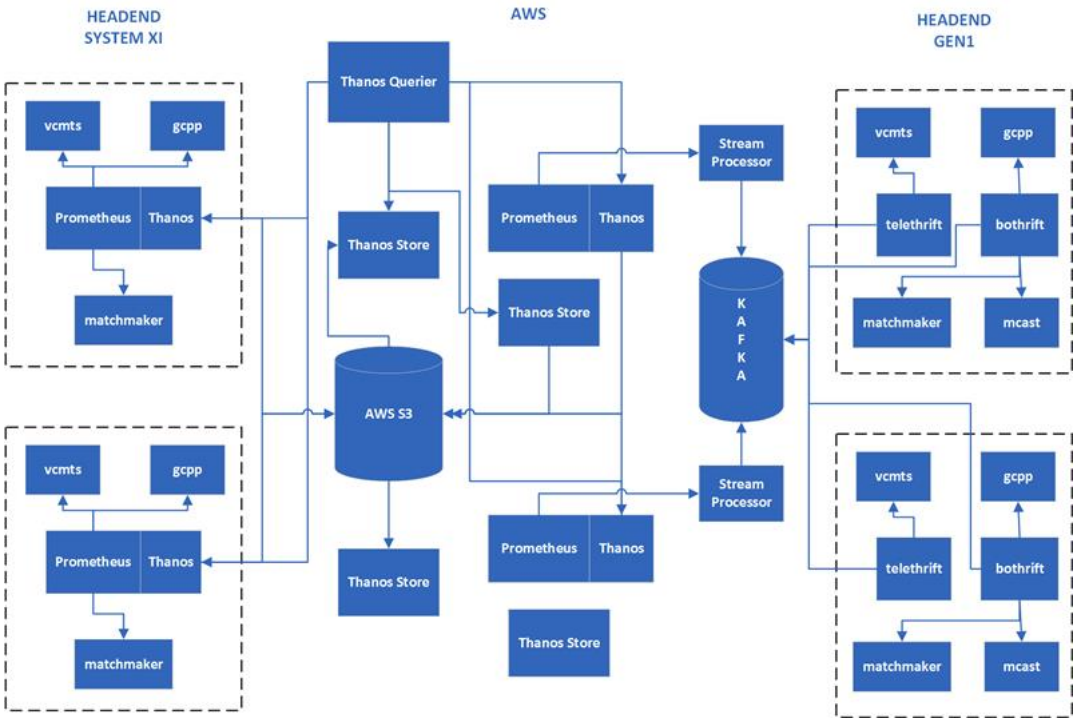

Figure 4 - New Telemetry Architecture

5. Containerization

Our containers run in Docker using Kubernetes nodes. We initially saw high CPU loads and high input/ output queue latency which often resulted in server cartridges hosting vCMTS worker nodes going into read-only mode. This resulted in having to cordon the read-only host, adding additional hardware (figure 5) to increase standby capacity and then fixing the read-only host during a maintenance window. Working closely with our server provider, HPE, we decreased latency of some Input/ Output requests that measured 25 milliseconds to around 0.8 milliseconds by changing Input/ Output timers in Kubernetes, changing queue sizes, and enabling multi-queue devices in Ubuntu Linux.

We also resolved Etcd issues that affected performance. Etcd is often referred to as the brains of the Kubernetes cluster. It is a distributed, reliable key-value store for the most critical data of a distributed system.

Lessons learned include regularly updating the Linux kernels and hardware drivers to take advantage of issues previously solved by the open source community. There has to be a balance between uptime and system maintenance. Also, it is important that your hardware provider understands how you are using their server (Kubernetes, Docker, Ubuntu, etc.) to optimize performance and assist you with choosing the best CPU and SSD for your application and alerting you regarding issues seen by other customers.

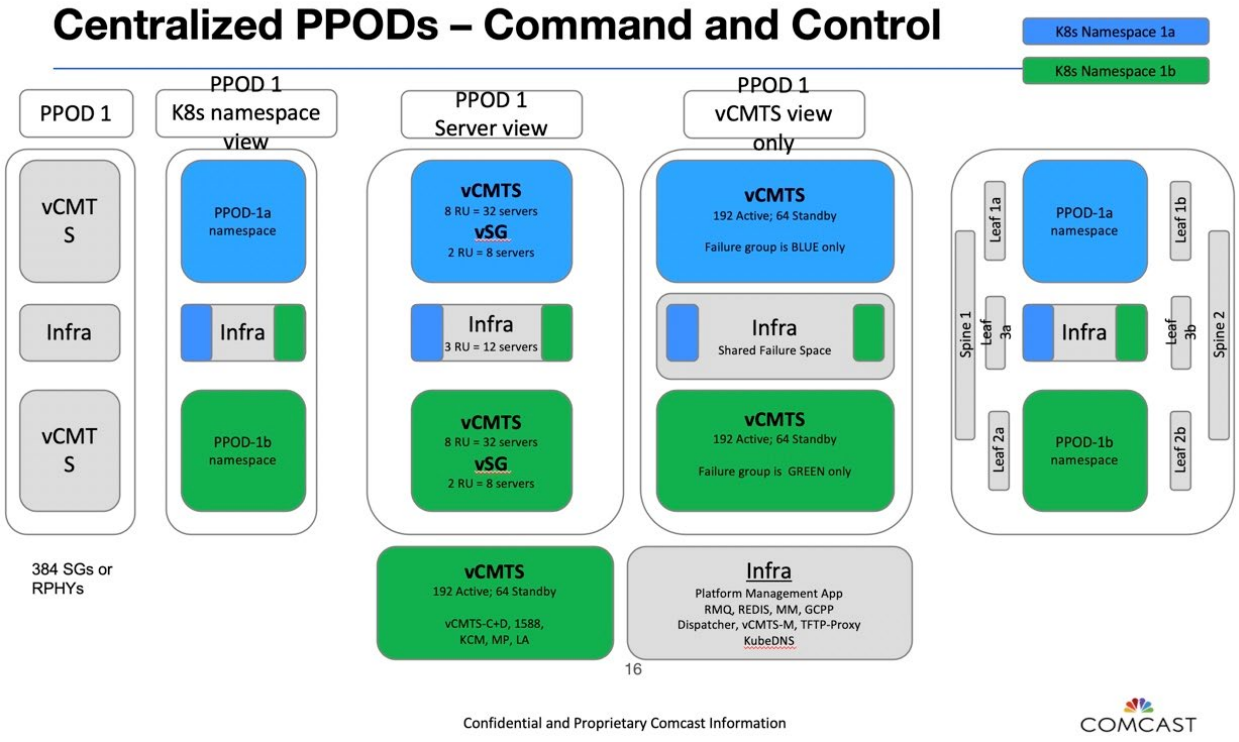

Figure 5 - PPOD showing Active and Standby nodes

6. Automation

We have had nothing but a great experience using Ansible for automation playbooks. Prior to Ansible, our team used hand-crafted scripts which were subject to human error. Other groups within Comcast use Jenkins and Puppet with equal performance.

7. Network Operating System

NGAN began using the Open Networking Foundation's (ONF's) Open Network Operating System (ONOS) in production during the first phase of DAA launched in May 2018. With this first generation, we realized significant cost savings in hardware while eliminating software licensing and software assurance costs. Using commercial off-the-shelf servers and white-box switches and working closely with ONF, we achieved limited scale (figure 6) by resolving a number of technical issues.

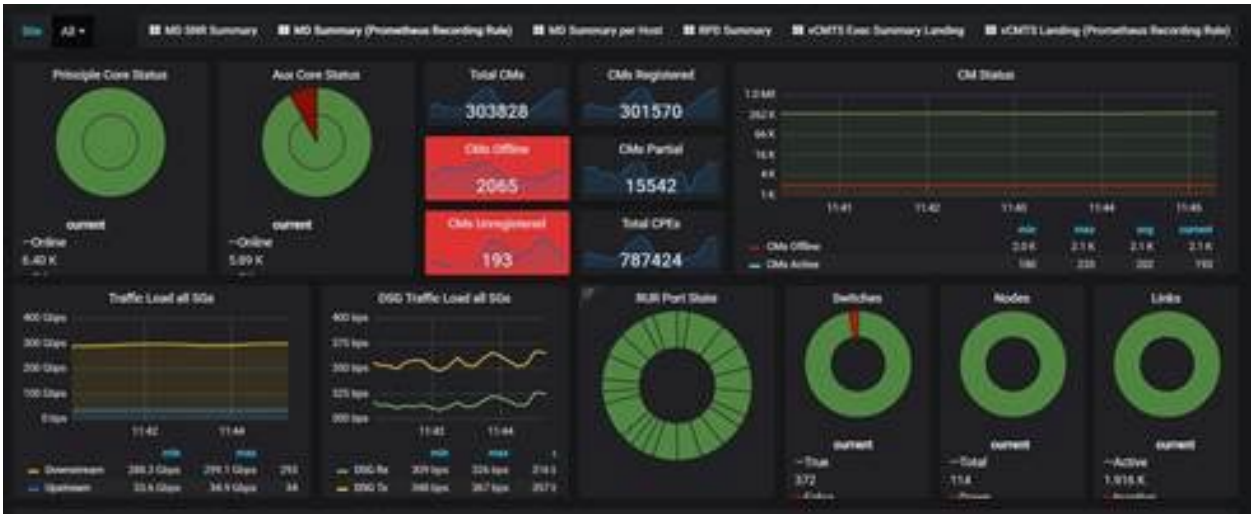

Figure 6 - Grafana Dashboard showing ONOS at scale

Progress included improving reliability during mastership changes, improving multicast handling performance, improving Atomix memory mapping and eliminating silent switch disconnects due to Open Flow socket closures. We achieved up to 40,000 cable modem scale in production by refactoring the route service.

While we had come a long way in terms of stability and performance improvements, ONOS clustering issues still persisted making it unlikely we could achieve our 99.999% uptime goal at 80,000 cable modems scale in time to support our deployment schedules.

Troubleshooting and redundant switch recovery at scale needed additional improvement. To operate at scale, additional features like headless operating mode, a more robust in-band management, handling internal queues , monitoring internal stores, and debugging issues at store level were also needed.

Despite the progress made and the significant investment in gaining ONOS expertise within NGAN, we made the difficult business decision in 2020 to move to a new, generation 2 architecture abandoning the open source network controller in favor of commercial network operating system (NOS)-based switches and a hybrid SDN Controller.

I believe we would have achieved our scaling and reliability goals, but we ran out of time to continue to optimize. We learned a lot in the process and contributed greatly to ONOS community with a number of key innovations. As a result of the progress that Comcast and its partners made commercializing ONOS, it is being evaluated by a number of wireless carriers in the 5G space.

8. Conclusion

Comcast was able to utilize OSS to build the access network with vCMTS and R-PHY. We continue to optimize the solution evaluating new hardware and software to improve reliability and scalability at reduced cost per subscriber. Telemetry, containerization and network controller changes were made to improve performance. A switch to a traditional Network operating system was made to immediately reach

scalability and up-time goals. We continue to evaluate new software (including OSS controllers) and hardware technologies to reliably improve cost per bit.

Abbreviations

5G	5th generation mobile network
AWS	Amazon Web Services
CLI	command line interface
COTS	commercial off-the-shelf (hardware)
DAA	Distributed Access Architecture
GNU	GNU's not Linux
HTTP	Hypertext Transfer Protocol
MSO	multiple system operator
NFV	network function virtualization
NGAN	Next Generation Access Networks
NOS	Network Operating System
ONF	Open Networking Foundation
ONOS	Open Network Operating System
OSPO	(Comcast's) Open Source Program Office
OSS	open source software
PTP	precision timing protocol
R-PHY	remote physical RF layer
SCTE	Society of Cable Telecommunications Engineers
SDN	software defined networking
SNMP	simple network management protocol
SSD	solid state device
vCMTS	virtualized cable modem termination system

Bibliography & References

https://www.fiercevideo.com/cable/comcast-backed-rdk-software-reaches-over-60m-devices

https://itnext.io/prometheus-for-beginners-5f20c2e89b6c
https://wiki.ubuntu.com/Kernel/Reference/IOSchedulers

https://medium.com/better-programming/a-closer-look-at-etcd-the-brain-of-a-kubernetes-cluster-788c8ea759a5

http://ospo.opensource.comcast.net/about/

Computer Ganga, "Advantages and disadvantages of command line interface", 2018. [Online]. Available: https://www.computerganga.com/2018/11/Command-line-interface.html [Accessed 6 August 2020]

