Dynamic Data Collection & Configuration Management

A Technical Paper prepared for SCTE•ISBE by

Rohini Vugumudi
Director, Software Development, Comcast
Comcast
Rohini_vugumudi@comcast.com

Co-Authors:

Hany Fame
Software Developer 5, Comcast
Hany_Fame@cable.comcast.com

Pardeep Singh
Software Developer 5, Comcast
Pardeep_Singh@cable.comcast.com

Zhen Lu
Sr. Software Developer, Comcast
Zhen_lu2@comcast.com

Table of Contents

1. Introduction

The modulation profiles for Cable Modem Termination Systems (CMTSs) have been historically applied manually, only changing with response to stimuli such as frequency impairments identified by field engineers or, worse, customers. This manual feedback loop is inherently slow, resulting in profile configurations that are limited by the impairments of the lowest common denominator on a given cluster of customers. With the advent of Data Over Cable Service Interface Specification (DOCSIS) 3.1 came the amazing and powerful ability to automatically adapt downstream profiles in near real-time leveraging machine learning and the Profile Management Application (PMA) concept. Tightening and automating the feedback loop allows for the recovery of previously wasted capacity, thereby making the entire network more efficient.

The authors of this publication have developed an implementation of the PMA concept that allows Comcast to manage the downstream and upstream environments efficiently at scale. Our current nascent architecture allows us to run a complete feedback loop every 6 hours; this runtime should only get better with further optimization of the Analytic Engine (AE) service, which is currently the bottleneck. Comcast can now optimize for the best performance, reliability and throughput in an automated and scalable fashion.

In this paper, the service architecture platform for dynamic data collection is called Genome, and it offers a key component of the overall configuration management system. Genome is responsible for the aggregation of data collected from cable modems and other customer devices, and the configuration management service is responsible for the application and validation of generated modulation profiles to their respective parent CMTSs. In particular, the details of adapting modern cloud computing tools to architect a reliable software solution for both downstream and upstream configurations are discussed. There are many details involved in the data aggregation, application of configurations, and validation of configurations, all of which are discussed.

2. Historical Context

The ability for cable providers to manage communications parameters for essential hardware has improved dramatically over the past several decades. The industry began with hardware that was statically configured or configured by physical manipulation. The invention of the DOCSIS specification allowed for configurable modulation profiles; however, profiles were statically defined by nature. In addition, while there is physically no difference between downstream and upstream channels, they have evolved differently over time due to how they have been historically allocated and used. This section will cover the evolution of both channel types and the difficulties in data collection and configuration management that arose because of this.

Data collection Limitations:

It is not a trivial task to collect the large amount of data required to create a continuous feedback loop for profile management at the scale of a national footprint. Previous attempts at creating a standard approach have been found wanting in terms of speed, cost, and maintainability. This section covers the limitations of common historical methods of data collection.

The most basic form of data collection has been to allow network engineers to manually record data in spreadsheets. This method is undeniably easy to implement but has many pitfalls. The most obvious is that the minimal feedback loop time is limited by human processing. In order to avoid potentially impairing customer experiences due to configuration changes, the sampling time must be reduced to as short a time frame as possible. Other difficulties that are associated with this method are those of standardization and the inclusion of human error. It is imperative that data sets are well sanitized before they are fed into a machine learning algorithm as input. Last, but not least, most solutions are plagued by scale. To consistently and reliably collect data at scale manually would require many network engineers, therefore increasing costs drastically.

As manual data collection can be ruled out due to practical limitations, the next step is to attempt to implement an automated collection scheme. Such a scheme must be able to scale in several different facets such as number of customer devices, number of CMTSs, and number of data points collected. Data collection from physical CMTSs such as the Arris E6000 converged edge router and consumer cable modems such as the Motorola surfboard SB8200 is limited to polling methods due to neither type of device having native software capable of pushing the required data types. One possible solution to scale is to manually segregate by CMTS or group of cable modems and put each partition onto its own Virtual Machine (VM). This is a practical solution given that the network does not change much over time; however, the demand for data services is growing at an unprecedented pace. Adding more data to feed additional features may result in the need to vertically scale VMs, while adding more CMTSs may require provisioning more VMs. While this is feasible, it would require considerable amounts of manpower in order to manage on a constantly changing edge network.

Configuration management Limitations:

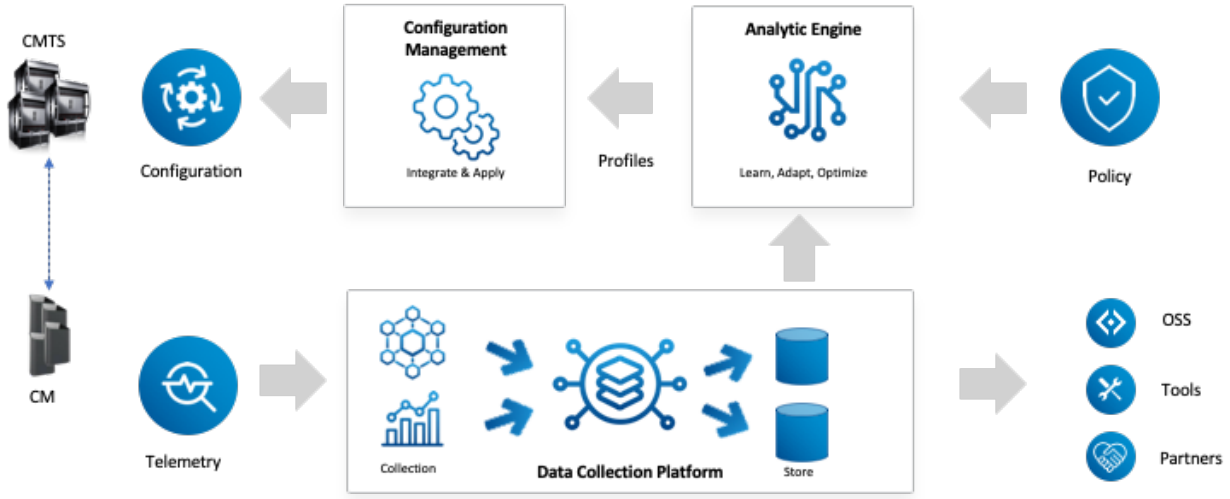
While the problem of dynamic data collection has historically been one of scale, the problem of configuration management has largely been one of standardization, process, performance, and risk. In the recent past, CMTS configurations were statically defined and rarely revisited for optimization due to manpower cost and risk. Even if one were to develop practical methods to collect the required data and use that data to generate optimized configurations, it would still be risky to apply manually said configurations as any mistakes made could negatively impact the customer experience. This fact has resulted in CMTS configurations that are generally very conservative in nature.

The tools and software available for configuration management in the past did not typically yield effective solutions. Most cable companies have, at some point, kept configurations tracked in spreadsheets. As in the case of data gathering, this is prone to human error and is limited in speed due to the human element. Previous solutions have also lacked standardized rules defining how conservative any given configuration should be. As a result, some areas may have more performance-oriented configurations at the cost of possibly degrading the experience of some customers, while others may have tended towards more conservative solutions that may have synthetically decreased the amount of usable bandwidth. Ultimately, the act of changing the configuration for a CMTS has not been something that could be done with the flip of a switch. Numerous validation checks must be completed both to adhere to lawful practices and to ensure that customer service is not impacted negatively. Any such system must also be agile enough to pivot in the case of process failure or external duress.

3. The Solution

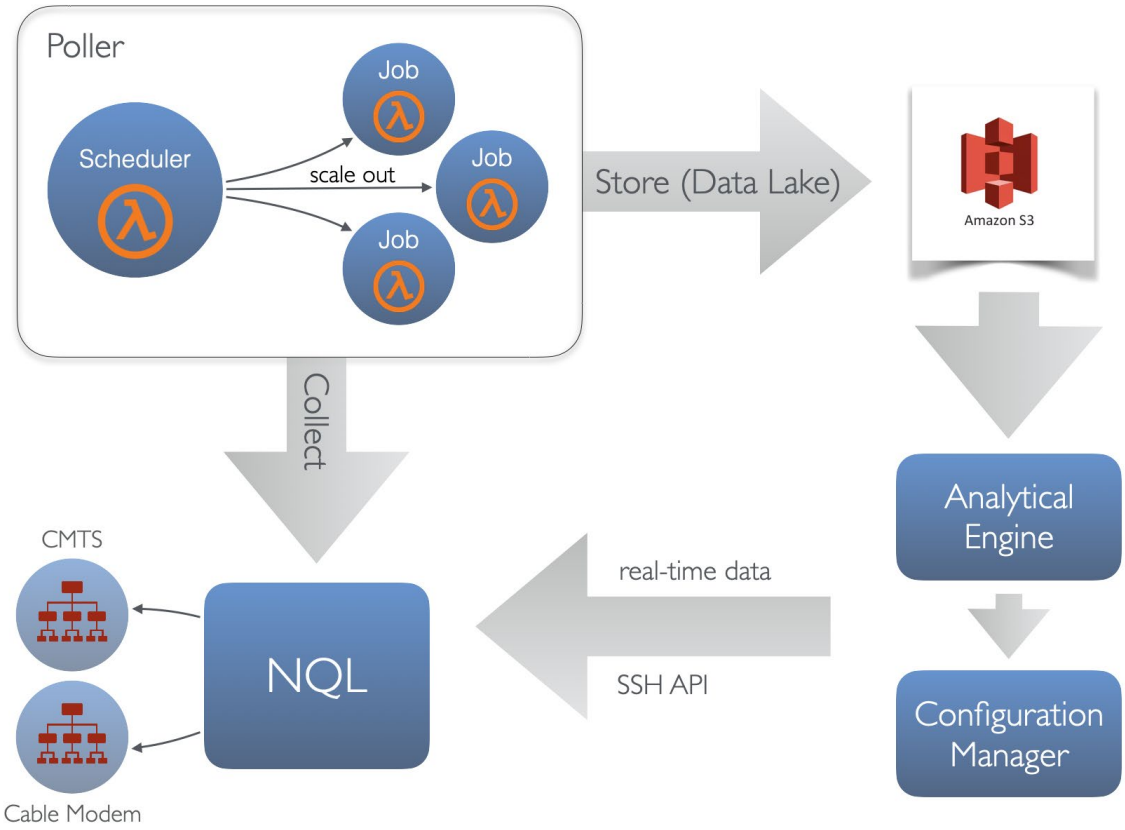
The previous section illustrates the need for a scalable, reliable, and automated approach for the activities involved in dynamic data collection and configuration management. This section discusses one implementation for such an architecture. It is important to note that this architecture is not limited in application to PMA, as it was built to be modular and support other internal processes around operational analysis and field support.

The main driver for the development of these services was to serve Comcast's PMA (Octave) efforts. Considering this, a brief refresher of Octave's architecture is given below before the focus is given to data collection and configuration management.



The first step of the Octave process is data collection, done by the data collection platform, which is internally named Genome. The purpose of this service is to poll both CMTS and cable modem telemetry data, sanitize it, aggregate it, and store it in a data lake for further use. In the case of Octave, the consumer of this data is the Analytic Engine (AE). For the purpose of this publication, this service will be treated as a black box, but interested readers can delve into the details in *A*

Machine Learning Pipeline for D3.1 Profile Management Harb 2019. The AE service generates optimized recommendations, then passed to the configuration manager service for translation into configurations. The configuration management service then does all the required validation checks before finally applying the CMTS configurations. This repeats in an infinite loop in order to keep the network configurations optimized.



3.1 Data Collection Platform (Genome)

The Genome platform was originally built to serve the Octave initiative but has grown into a platform in its own right since inception. The main purpose of Genome is to actively poll and cache data from devices such as cable modems and CMTSs in a scalable and configurable way while offering consumers the ability to analyze the cached data or get live data in a seamless fashion. This positions Genome to be the sole data provider for cable modem and CMTS data, eliminating inefficiencies around oversaturating CMTSs or cable modems with connections and over fetching duplicative data. In the context of Octave, Genome is responsible for collection, standardization of data required for the creation and management of modulation profiles. Genome

data collection is utilized in the analysis of Proactive Network Maintenance (PNM) activities as well as other field support analysis tools internal to Comcast in addition to Octave.

Genome is made up of two layers, the poller layer, and the query layer. A "poller" is simply a piece of software that is responsible for scheduling, collecting, and standardizing a set of data from devices. Pollers are built to be lean, modular, and extensible. The query layer for Genome is called Network Query Language (NQL), which exposes a declarative API service through which consumers request live or cached data. All pollers are primary consumers of NQL to collect live data, while also offering external consumers to do the same. NQL's goal is to offer a declarative abstraction layer for edge network devices, allowing consumers to query using standard HTTPS instead of SNMP, TFTP, and various other network communication protocols.

Genome requires a master list of all CMTSs to poll as well as a configuration for the polling cadence for each data property. When a CMTS gets added to the network, it should be added to this master list. In the case of Octave, this master list is automatically updated through Comcast's deployment ticketing system. On the next poling cycle, Genome will discover the newly added CMTS and request a list of all cable modems it serves. Afterwards, Genome maintains a cached list of all cable modems associated with the CMTS. The list is the source to get the OID data from all saved devices. The polling could be done at any configured frequency or on-demand.

In addition, Genome must ensure that data collection can scale when CMTSs are added over time such that data collection and aggregation can be achieved in a given polling window. It must also ensure that the ingested data is validated and cleaned before it is cached. As the amount of data is large, especially in the case of cable modem data, Genome must also manage data retention policies in order to reduce cost.

The following are some examples of data points that Genome collects:

- OFDM Channel
  - OFDM channel width
  - Subcarrier width
  - Start frequency
  - Active & excluded regions
  - Position of PLC channel
- OFDM Subcarrier
  - Modulation efficiency per subcarrier
  - Subcarrier type
- CMTS
  - Make
  - Model
  - Hardware
  - Software version
- Telemetry
  - MER
  - FEC
  - Traffic

NQL's journey began in the early days of Octave development. The goal was to create an API service which abstracts away all different protocols around networking and let end users interact through HTTPS. The first iteration came in the form of a REST API which accepted OIDs parameters and returned the output through HTTPS. While this iteration ran in productions for several months, several shortcomings were brought to light. For example, use cases involving many data points per CMTS, necessitated making multiple requests to the API. Each request would connect to the CMTS, which resulted in opening and closing sockets many times over multiple requests. Furthermore, the work of encoding and decoding large amounts of JSON data was repeated for each request, degrading the overall performance of the service through repetition of work. It became clear that the architecture needed to evolve in order to overcome these shortcomings. Indeed, with the vision that, Genome, and by extension, NQL, would grow to encompass more and more, it was clear that a technology would have to be chosen to all for incrementally extending the codebase without causing breaking changes at every turn. GraphQL was a natural option that allows data schemas to easily evolve and can handle complex relationships between data sets. NQL was born.

As NQL offers an abstraction layer for connectivity to both cable modems and CMTSs, it must therefore manage all writes as well as reads to each network device. In order to accomplish such a feat, it supports both IPv4 and IPv6 communication protocols. NQL is built primarily using the GraphQL specification and has been optimized at node level. In some cases, consumers may require a request-response type handshake, where the consumer keeps a socket open until the requested data is returned. However, many queries may be long running and it may not be practical or possible for a consumer to keep a connection open for the duration of the process.

Nodejs Historically, the team used Nodejs as a declarative jack-of-all language. However, in this use case, performance is king. As Golang is routinely touted as a performance-oriented language geared towards networking applications, it was a simple choice after orchestrating some internal benchmarking. Simple GraphQL APIs were built in both languages to test a small fraction of our total scale. The goal was to see how many VMs would be needed to handle the same amount of work while profiling the performance of each VM by measuring CPU and Memory usage. As Nodejs is single-threaded language, it is more difficult to saturate a larger VM without adding complexity. To keep it a simple apples to apples comparison, small VMs were used for both languages. After few weeks of tweaks and testing, the conclusion was that Golang is around 3-4 times faster in handling the same work as Nodejs for this use case. This was not a perfect apple to apples comparison since many variables are in play here. Particular to note is the reliance on third party open-source packages, whose performance is out of our control. Another thing that was found is that Golang was very consistent in its response times and overall API metrics, while the API built using Nodejs would suffer spikes in response times. This is likely due to node's single-threaded nature, which makes it not well suited for CPU intensive tasks that block the main execution thread. Golang was the clear winner in this exercise and was used going forward.

NQL's SSH feature allows consumers log into a supported remote device through HTTPS rather than SSH. This allows NQL to abstract away details around authentication, authorization, and managing the underlying SSH connection. Users are able to connect to a host, run multiple commands, and get output back for each command. NQL uses GraphQL to define the API, which

allows developers to develop powerful features and evolve our API without breaking the world. NQLs SSH is a simple layer around secure shell, all the logic around what commands to run in what order and what to do with the output is handled by the client itself. For example, CM can apply configurations to a CMTS and then analyze the output to see if everything went well. All this is done through HTTPS using NQL. NQL has evolved from a standalone service running on a EC2 VM to where it could be leveraged anywhere within our codebase and able to run in any container or lambda alongside our codebase.

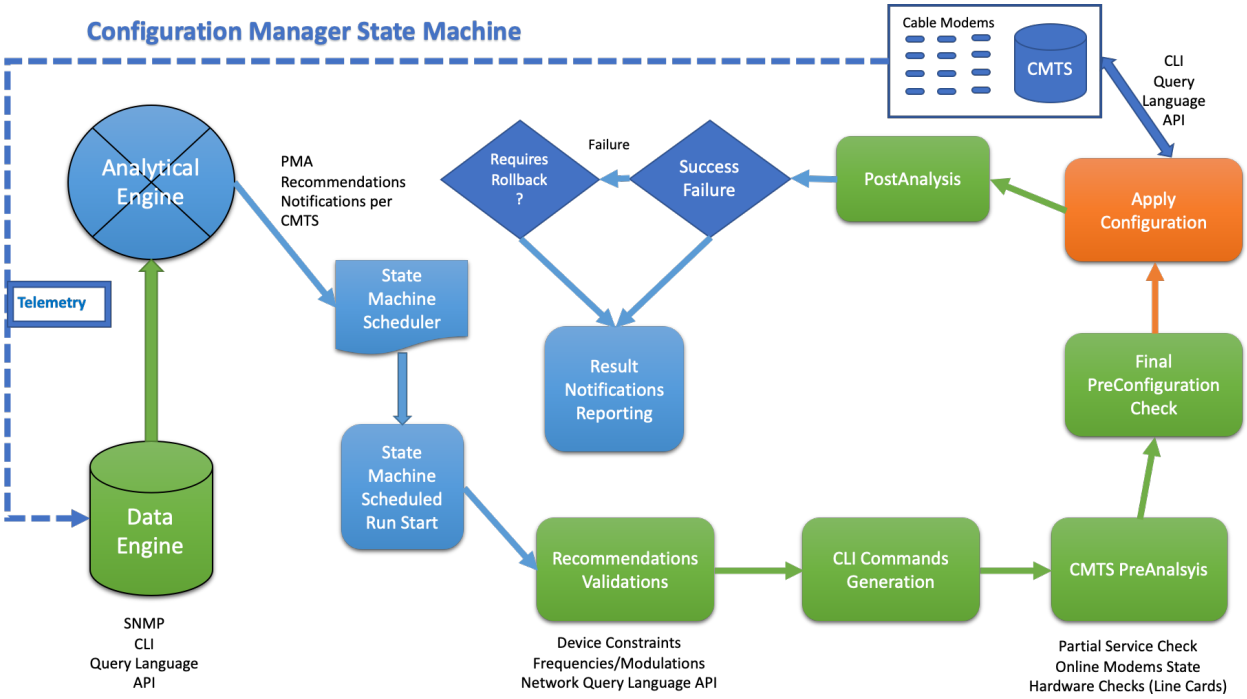
SNMP v2/v3 capability

- Genome by default uses the polling on using SNMP V3, and if the device is not V3 enabled, the fall back option is V2.
    - Get public and manager key from cable modem
    - Use manager key and V3 security name as input to get vault secret
    - Use cryptographic hash function on vault secret and public key and compute auth priv keys.
- Collect the SNMP data and stream to Kinesis which will be used by Analytical Engine from PMA and other consumers of the data as need basis.
- AE consumes the data and suggests the profile which will be sent to configuration manager, which further defined below how the process works and how the recommended profile has been applied to using the configuration manager

3.2 Configuration Manager

The Configuration Manager (CM) service is a scalable state machine application which is able to generate and apply modulation profiles to CMTSs based on suggestions provided to it by the AE service. It was designed to enhance DOCSIS3.1/3.0 downstream/upstream capacity and correct for RF noise impairments without manual interaction. Unlike the Genome service, it is currently not a standalone service and requires input from the AE service to function.

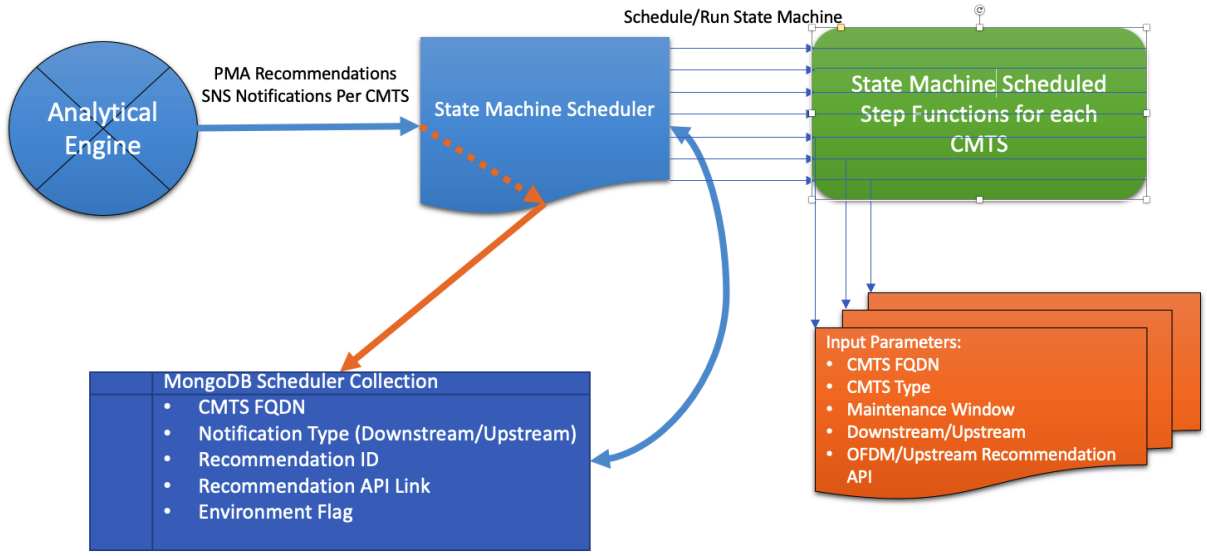
Detailed Work-flow description of Configuration Manger:



- The Configuration Manager State Machine receives a decision recommendation in a continuously timed workflow from the analytical engine. The recommendation can be to adjust or change the DS OFDM profile modulation (frequencies, QAM modulations, default modulations for each subcarrier) or can be to adjust or change the upstream profile modulation. These recommendations are based on telemetry feedback from the data engine. The recommendations should be universal in nature and are not tied to specific type of CMTS machine.
- The application schedules the state machine runs for the particular CMTS according to a predefined schedule.
- At the time of state machine operation, the CM application validates the recommended changes, conducts numerous pre-analysis checks on the CMTS using telemetry information, verifies the proper current state of the machine, and validates the overall system health.

- If the validations pass, then commands will be generated based on the AE recommendation, the type of the CMTS device, and whether it is a downstream or upstream modification.
- Before applying the commands on the CMTS machine, other validation checks are made.
- Modification to the CMTS will be made and tracked for any failures during post analysis, and the system decides if any rollbacks are required or not. The system will finally report the result of the state machine run for the CMTS, which can be tracked through the database, reporting, and dashboards.

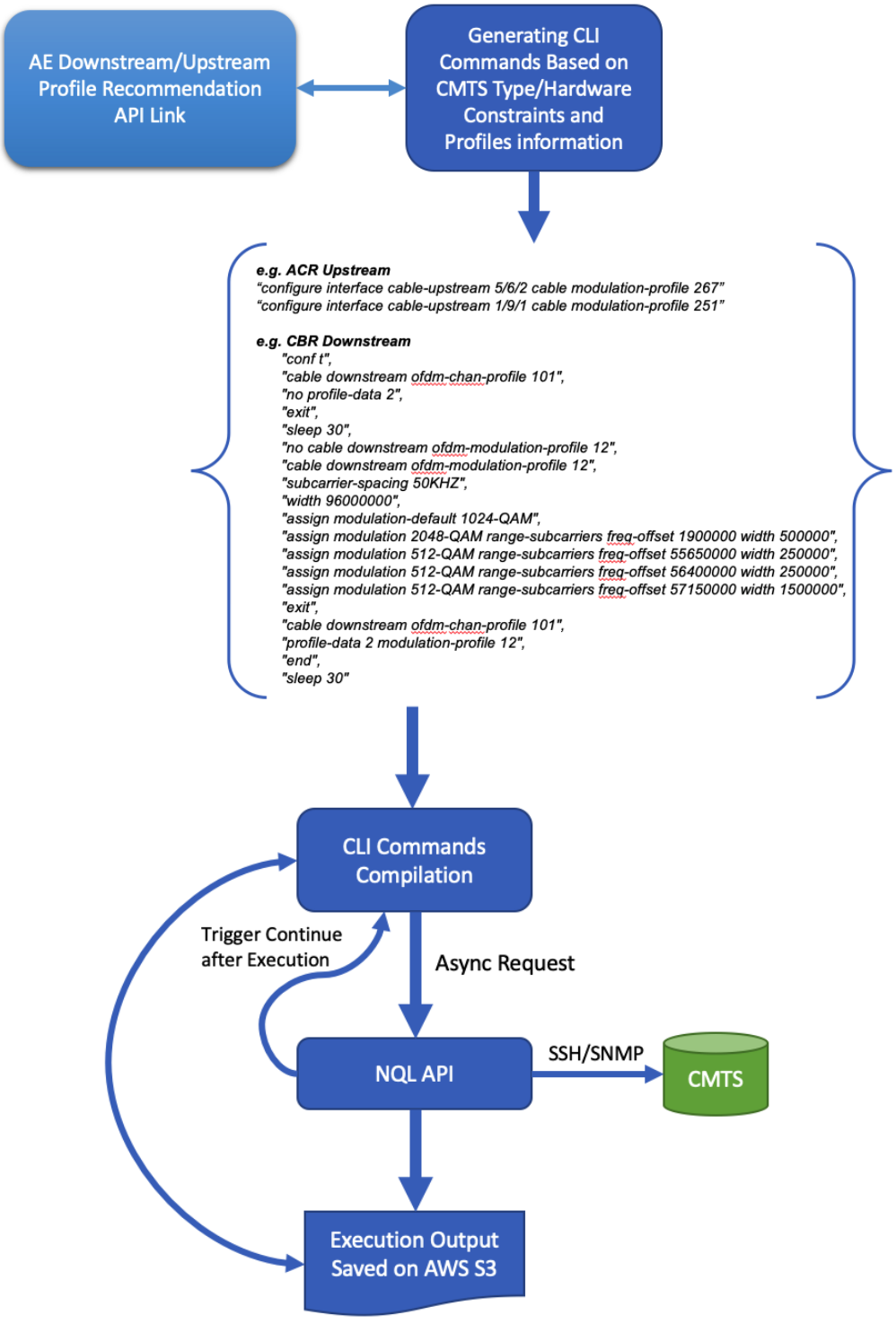
3.2.1 Configuration Manager State Machine Scheduler



3.2.2 CLI Commands Generation and CMTS Device Configuration

- In the CLI command generation phase, the function generates arrays/objects containing list of the commands which will be executed in order on the CMTS device.
- The method with which the commands are generated depends on the type of device, as each device uses its own set and way of CLI configuration. It is also based on the profile recommendation received from AE. For example, modulations and frequencies are calculated in real time based on OFDM segment start/stop, modulation bits provided, and other parameters such as lower and higher OFDM frequencies spectrum on a particular interface.
- The commands are saved in AWS S3 for the current step function to be executed during apply configuration phase in the state machine.
- In the apply configuration phase, the application uses the S3 file data object to compile the list of commands to be executed by NQL API

- An Async request is then sent to NQL API to execute the commands, the state machine waits a trigger back from NQL API for the full execution output for all CLI commands sent.
- The CLI commands output is analyzed for any errors, failures, or timeouts during execution on the CMTS.



3.2.3 Configuration Manager State Machine Supports and Uses the Following Technologies:

- AWS Serverless Step Function Architecture:

Each step of the state machine is handled with separate lambda function. The result of each lambda function dictates the next step of the state machine

- AWS DynamoDB
  Used for device inventory collection
  Environment flags
  Device Constraint Information
- AWS S3
  Used for dynamic command generation
  Pre/Post Analysis Device output
- MongoDB:
  State Machine Scheduled Collections and Reporting
- NetScout API for Analysis of 911 Calls in Progress on CMTS
- ServiceNow API for Change Management
- ASYNC Apply Configuration and Device Checks

4. Challenges / Discoveries

4.2 Genome

Major improvement of genome is to move from the Initial old solution with physical servers to a much more efficient and performance oriented solution.

Iteration # 1 Moving from old solution to Improved process – Tech stack change
Financial Implications:

- Resources
  - Had to come up with new resources who suits and understands new technology
- Infrastructure
  - Build most scalable and maintainable solution
  - Select and move to the technology stack which is cost effective
  - Secured solution
- Standardization of network
  - Support all different versions and different OS's differences
  - Solution which supports, evolving network with minimal changes
- Data reliability
  - Techniques in place to make the data reliable
  - Ways to track the data and blockers through dashboards

Political implications

- Sell the business case and architecture solution with the right reasons
- Show the brighter side of the new solution than seeing reasons how old solution didn't do
- Take the limitations had before as requirements
  - Performance improvements
  - Data Reliability
  - Dashboards
  - Alarms and notifications to act on

    - SLA's in place to support downstream systems
- Long term vision of scalability and reliability
    - Minimal investment
    - Minimal efforts in upgrades
    - Much more configurable

Iteration # 2 Moving improved process to stable platform

Considerations around the Stable platform:

- IPv6 protocol compatible
- Migrating from SNMP v2 to V3
- Supporting both SNMP V2 and V3
- All new solution around USM Key store

Advantages presented with the new stable platform:

- Cost effectiveness
- Reliability of network connectivity
- Security enhancements
- Enterprise level USM key store solution
- Long term vision of scalability and reliability

4.3 Configuration Manager

Most of the challenges we encountered was related to CMTS hardware limitations, configuration or CLI limitations, and cable modem bugs/firmware issues which cause modems not to behave as expected with upstream or downstream profile modifications.

Iteration #1 Arris ACR E6000 Downstream:

- CLI Timeout Enhancements
- 911 Check Enhancements
- Validation Enhancements
- Working on improving overall state machine work flow for scalable deployment

Iteration # 2 Arris ACR E6000 and Cisco CBR8 Upstream:

- Cisco CBR8 implementation was challenging, during testing we found out that after modifying upstream channel profile IDs, the SNMP validations were incorrect and profile were not aligned, after significant troubleshooting, we realized that CBR8 requires upstream channel to upstream controller mapping to modify proper assigned upstream profile IDs.

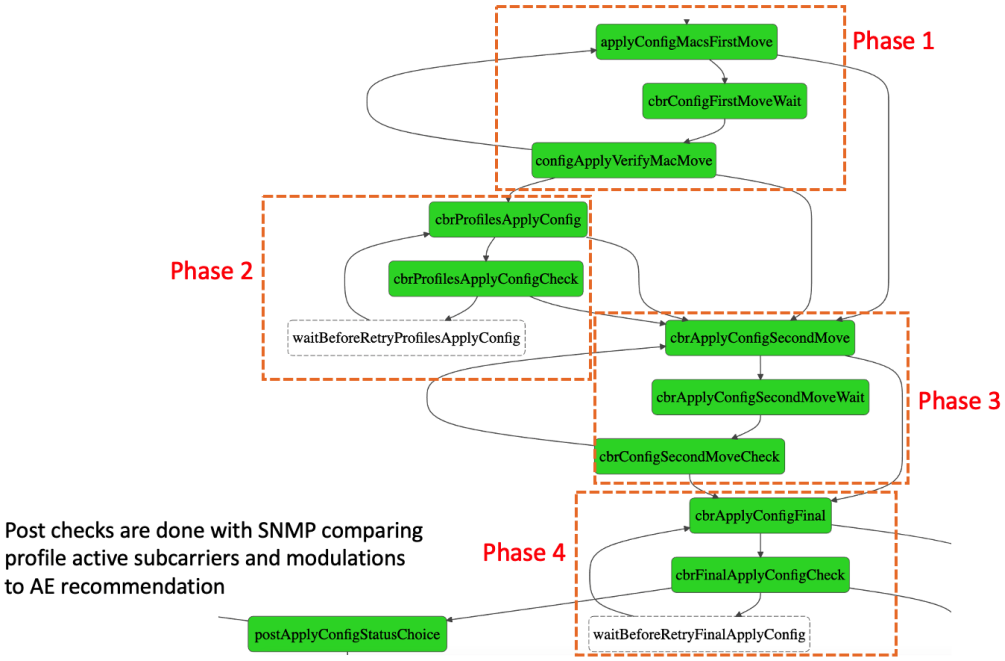
- Arris ACR E6000 encountered significant issues where cable modems go into upstream partial service, which was concluded to be a bug on the E6000 which was resolved with a firmware update, and also bug in few of particular types of the Motorola modems SB21x and SB61x which we had to work on some workarounds and run debug commands to move the modems out of partial service, as well as Zoom modems required firmware upgrade which was required to prevent them from going completely offline.

Iteration # 3 Cisco CBR8 Downstream:

- Cisco CBR8 implementation required almost a complete new design for the state machine, since CBR8 doesn't support modifying OFDM profiles globally on the cable channel, each DOCSIS 3.1 modem must be moved from a particular profile and to another one in order to modify the profile that this modem is currently using, then moving the modem back to use recommended profiles, this means significant number of CLI commands had to be generated and executed on the CMTS, one thousand modems requires 3000 CLI commands along with other profile modification commands
- We were able to achieve this by splitting the commands and apply configuration into four phases in the state machine, with each phase requiring validation check, we also used ASYNC apply configuration to limit timeouts, maximum time for each phase is about an hour to hour and half which should be fine for implementation on a fully loaded CMTS with about 3-4K DOCSIS 3.1 cable modems.

1. **Phase one:**
   Shutting down internal PMA
   Locking cable modems to control profile 1 and moving service flows to profile 1 (only for profiles requested by AE)
   Verifying all modems are now locked to profile 1 after waiting few minutes, if any modem(s) still stuck on other profile, CM will put the profile into ignore list and will not modify its OFDM subcarriers in phase two
2. **Phase two:**
   Applying OFDM profile changes (subcarriers and default modulations) and verifying configuration lines are applied
3. **Phase three:**
   Unlock all modems in phase one from profile 1 (moving service flows from profile 1)
4. **Phase four:**
   Turning internal PMA back on
   Do OPT commands on all modems from phase one to speed up process of modems to use appropriate profiles

Phase 3 and Phase 4 will run regardless of Phase 1 and Phase 2 result, this is basically rolling back the modems to their original state and letting the CMTS assign profiles, this must be done even if we have failures in phase 1 or phase 2, or modems will stay locked to profile 1 and will stay locked for subsequent PMA runs unless manually unlocked.



5. The Future

We have built a platform which is scalable and maintainable. Where do we do we go in the future? What are the possible improvements and upgrades that we could add to make the platform work even better for the organizations? How could we use the data that is being collected beyond the initial reason it started?
What else can we do with the configuration management platform to go beyond applying configurations? How could we improve our processes?

We will now share a couple of ideas and architectures for our future work in the areas of data collection and configuration management. Irrespective of the data collection side or the configuration side, major focus for future for the platform as a whole would be:

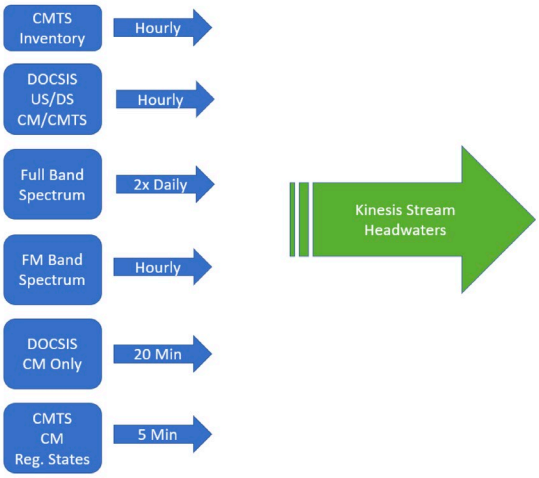
- Vendor agnostic
  - Customizable and configurable metadata for data collection points
  - Irrespective of the vendor
  - Configuration application API
  - Configuration template management
  - Upgrades to all kinds of OS upgrades and patches

- Vendor specific
  - Standardized and easy update and upgrade of vendor specific data points

- Upgrades to all kinds of OS upgrades and patches
- Easily updatable configuration templates methods

Data Collection: (Not Limited to)

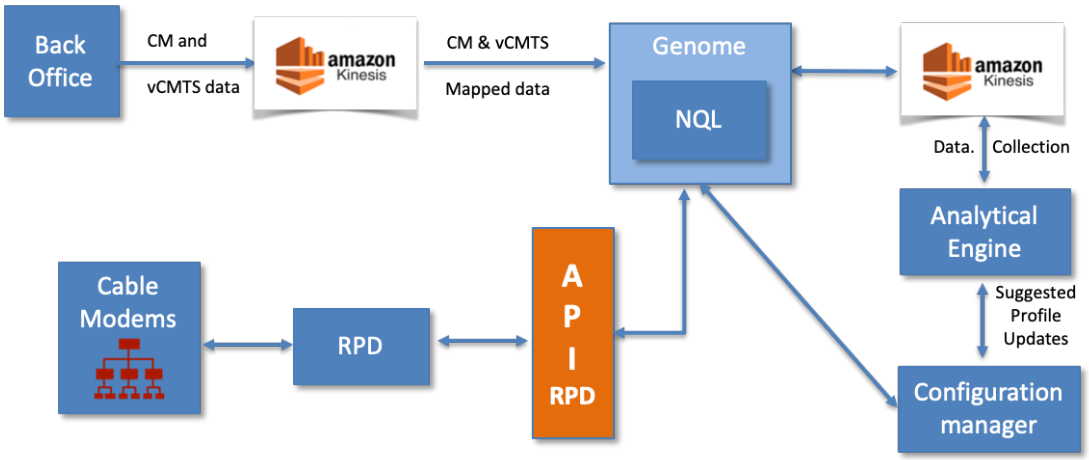
- Consolidation of the data collection across the organization
- Reduce the number of pollers connecting to devices
- Improve the performance of the devices with less pinging
- More optimizations for better performance
- Expanding the data collection beyond CMTS and CM to power supplies, RPD's ,vCMTS
- Expanding the data collection specific with adjustable frequencies



Configuration Management: (Not limited to)

- Standardization of the configurations across organizations make deployments easy
- Separate it into a standalone service that so that other services can take advantage of the validations and checks it employs
- Standardize managing configuration templates
- No manual errors in initial system configurations
- Site specific customized configuration templates
- Vendor agnostic platform
- Easily adaptable, manageable, expandable

Genome/Config Management Architecture future for vCMTS Proposed for Octave on vCMTS platform



6. Conclusion

Reliability, Manageability, Consistency, Usability, Performance, and Scalability are the most popular buzz words in the market, and they are now becoming the base necessary requirements for the creation of any software product/platform. Keeping that in mind, this study provides valuable insight to the ongoing evolution of creating cutting edge technology solutions for the Cable Industry.

In addition to a review of the past and present, the paper also demonstrates the benefits of using modern tools and infrastructure components. In the opinions of the authors, reliability in any kind of service/platform is the most critical component to create customer satisfaction. A commitment towards the better Customer Satisfaction always results in the best product.

Abbreviations

CMTS	Cable Modem Termination System
CM	Cable Modem
US	Upstream
DS	Downstream
NQL	Network Query Language
ISBE	International Society of Broadband Experts
SCTE	Society of Cable Telecommunications Engineers
USM	User-based Security Model
Auth	Authentication
SNMP	Simple Network Management Protocol
Genome	Data collection platform Name

Bibliography & References

A Machine Learning Pipeline for D3.1 Profile Management, Harb, Ferreria, Rice, Santangelo, Spanbauer, 2019

