SCTE CABLE-TEC EXPO 22
SEPTEMBER 19-22 • PHILADELPHIA, PA

2022 Fall Technical Forum
SCTE • CABLELABS • NCTA

Scaling DAA: Automated Network Health Check for vCMTS Platform

A Technical Paper prepared for SCTE by

Marissa Eppes
Data Scientist
Comcast
1800 Arch St, Philadelphia, PA 19103
Marissa_Eppes@comcast.com

Ilana Weinstein
Data Scientist
Comcast
1800 Arch St, Philadelphia, PA 19103
Ilana_Weinstein@comcast.com

Matthew Stehman
Senior Data Scientist
Comcast
1800 Arch St, Philadelphia, PA 19103
Matthew_Stehman@comcast.com

Table of Contents

Title **Page Number**

List of Figures

List of Tables

1. Introduction

As Comcast progresses through Gen2 of the Distributed Access Architecture (DAA) initiative, "automating everything" is paramount to the continued growth of the footprint. With the exciting milestones achieved in Gen1, which took the first DAA customers online, came a laundry list of operational improvements needed to realistically achieve the desired scale. The common theme among these needed improvements? Automate. So far, Gen2 has made considerable progress in this regard, automating everything from individual virtual cable modem termination system (vCMTS) cluster standups to software and network changes, to incident detection and mitigation (Krishnamurthy & Medders, 2021).

Since virtualization is the name of the game in the world of DAA, the vast majority of vCMTS maintenance and upkeep is achieved with cloud component software updates, specifically component configuration changes. Achieving near total automation involved transitioning software upkeep over to a DevOps approach, with which changes to individual clusters are managed with cluster-specific continuous integration/continuous deployment (CI/CD) pipelines. With this setup, a single Git commit indicating a configuration change to a specific vCMTS component schedules the entire deployment process, which kicks off a cascade of automated events, including but not limited to silencing alarms, microservice performance checks, the actual software deployment, and a network health check to ensure no customer impact. The goal in using this strategy to orchestrate software deployments is to eliminate as much human interaction as possible, as even the slightest human error introduced at a single step can have monumental impacts to the cluster configuration and network performance down the line and can lead to complex outages and delay feature enhancements and releases.

During software updates, preserving or improving the customer experience is of the highest priority. In Gen1, network health was checked manually; an operator would verify no customer impact by checking telemetry dashboards and manually flagging any signs of service degradation. Not only is this practice not scalable but subjecting this critical process to the risk of human error is undesirable as DAA scales and the customer base grows. Therefore, automating and optimizing network health monitoring surrounding software updates is of utmost importance. This paper highlights a data-driven approach to developing and productionizing an automated network health check for use in vCMTS deployment CI/CD pipelines as part of the Gen2 DAA initiative.

1.1. Problem Statement

As the number of software updates needed to maintain and scale the DAA footprint increases from tens to hundreds to sometimes thousands per day, it is necessary to not only have these automated processes in place, but to ensure that they are optimized to near perfection. With millions of customers already converted to DAA, automated network health monitoring is one of these essential processes, given that software updates can sometimes cause unintended side effects on the network, resulting in a degraded customer experience. Software-related service degradation might manifest as anything from interruption of service to poor traffic throughput, to partial utilization of network capabilities, and more. Even though most software updates pose little risk to the customer experience, catching these occurrences when they do happen so that quick mitigative action can be taken is considered to be a critical capability. Therefore, there is a need for an automated and dependable tool for post-deployment network monitoring, which can validate that software updates do not degrade service for the existing customer base *or* flag the occasional software updates that do.

To deliver true value and support the goal of total automation, this tool needs to be compatible with the vCMTS deployment CI/CD pipelines and to deliver highly accurate results, alerting the ops team only

when there is definite service degradation detected following a deployment, while simultaneously maintaining a level of sensitivity as to not let any undetected service degradation slip through the cracks. Additionally, to adhere to target maintenance window timeframes, this tool must determine and deliver a decision on the state of the network back to the CI/CD pipeline within a matter of minutes following the deployment. Last but not least, if service degradation is detected, the tool must provide a meaningful summary detailing the reasons for service degradation so that an operator can take the appropriate action.

1.2. Solution

To address this need, the data sciences team has leveraged the rich network telemetry available within the DAA system to develop a network health decision engine made available as an application programming interface (API). When called, the API queries cluster-specific telemetry metrics live from a time series database and performs a suite of algorithms to assess the health of the network. The API response flags any incidental impact on already-live customers and alerts an operator within a matter of minutes.

The data sciences team has collaborated closely with the DAA engineering and ops teams to identify relevant network key performance indicators (KPIs)—all in accordance with Data Over Cable Service Interface Specification (DOCSIS®)—to check as part of this process. The application performs *i.* a pre-check to gather a baseline measure of network KPIs prior to a software update and *ii.* a post-check to draw comparisons among KPIs and ensure that the network remains in a healthy state following a software update. The post-check assesses the state of the network instantaneously by comparing instant post-deployment KPI readings to either instant pre-deployment or historical KPI readings. The API is designed specifically to integrate with the vCMTS software deployment CI/CD pipelines and is robust enough to offer a one-size-fits-all solution to all clusters, regardless of configuration differences, number of customers, differences in downstream topology, etc. The API is also compatible with a variety of software deployment types, regardless of the target component and predicted risk level.

The API is currently integrated and running in a production environment. The application is invoked during each software update for a variety of vCMTS components ranging from server builds to configuration updates to operating system upgrades. As measured in recent analytics, the application is invoked, on average, ~400 times per day and triggers alarms on ~2.5% of invocations. This paper will take a deep dive into the methodology used to develop the tool and tune the rule-based algorithms, present performance metrics, discuss lessons learned, and briefly touch on relevant future work.

2. Background

2.1. DAA Topology

While DAA offers a technologically progressive means of providing service to customers, the distributed architectural setup is rather complex. The access network contains a variety of physical and logical components ranging from headend Kubernetes servers to a series of leaf-spine switches to downstream digital nodes all the way down to the customer premises equipment (CPE). When one or more of these vCMTS components undergoes a software update, the deployment CI/CD pipeline performs a series of checks on the entire cluster footprint. These checks can be grouped into two general categories: *microservice performance* and *network health*. Because this paper focuses on the latter, the intricacies of the DAA architecture and the vCMTS cloud environment will be outside the scope of this paper. However, several past papers cover these topics in detail, namely *Distributed Access Architecture Is Now Widely Distributed - And Delivering On It's Promise* (Howald et al., 2021) and *Node Provisioning and Management in DAA* (Gaydos et al., 2018).

To understand how the automated network health check is performed from a customer impact standpoint, three topological entities must be understood: *i.* the physical point of deployment (PPOD), *ii.* the remote PHY device (RPD), and *iii*. the cable modem (CM). Figure 1 portrays a simplified view of relevant DAA topology.

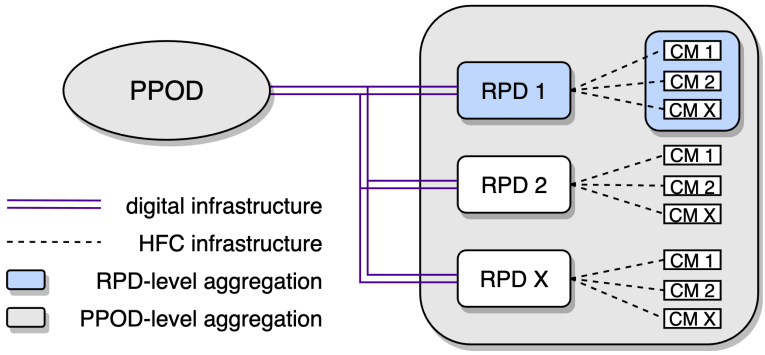

Figure 1: A simplified view of DAA topology

2.1.1. The PPOD

Located at each primary headend, PPODs comprise the actual servers on which DAA software is deployed. PPODs and vCMTS clusters are often referenced interchangeably and, in theory, describe the same technology. However, usage of one term versus the other depends on the context in which the technology is being discussed; a PPOD can be thought of as an abstract deployment unit, whereas "vCMTS" is often used in reference to the physical cluster hardware. Given that this paper discusses DAA from a software standpoint, we will largely use the "PPOD" naming convention going forward.

As Krishnamurthy and Medders discuss, clusters are often spun up with configurational differences, "each with their own slightly different personalities" (2021). Given that any two PPODs might be configured differently, the PPOD is the highest-level architectural component on which it makes sense to perform a series of automated checks and aggregate results. Even when the entire DAA footprint needs to undergo a particular component update, performing PPOD-level checks eliminates any risk of confounding configurational differences into the equation. As such, all DAA deployment CI/CD pipelines are kicked off at a PPOD-level, and subsequent checks are intended to indicate how a particular PPOD "personality" fairs with any given software update. From the network health perspective, this entails checking for any service degradation or incidental impact experienced by customers downstream of the PPOD of interest.

2.1.2. The RPD

Designed to bring digital data transmission as close to the home as possible, RPDs sit on the very edge of the access network and comprise the gateway between the digital system and the hybrid fiber/coax (HFC) network, which eventually reaches the home. RPDs are the most downstream digital component of the access network and can be subject to software updates themselves. Like PPODs, RPDs can have configurational differences and/or come from different vendors, creating variety among them. Therefore, RPD-level data aggregations provide diagnostic value when conducting the network health check. The strategy of grouping together customers downstream of each RPD and analyzing each of these customer subsets separately provides a more thorough network health validation and, if service degradation is noticed, helps the responding operator to determine if the source of the issue lives at a particular RPD.

2.1.3. The CM

The CM, which is often used interchangeably with "CPE", is the final topological entity involved in the network health check. Individual CM metrics are often the fundamental units used to derive data-based network health algorithms, and all CMs subscribed to the PPOD-of-interest are taken into account during the assessment. However, with millions of CMs already dispersed across the DAA footprint, it would be impractical and disadvantageous to collect, analyze, and report on telemetry data for each and every CM. In analyzing CM metrics, aggregations at both the PPOD- and RPD-levels deliver a more meaningful and statistically robust measure of how the customer experience is fairing across the footprint following a software update. Additionally, if service degradation is detected, this aggregation strategy offers a diagnostic advantage in that it can pinpoint a lowest common ancestor for problematic CMs, allowing for quicker isolation of the issue. For these reasons, almost all algorithms operate at either the PPOD- or RPD-level. The only exception to this aggregation strategy is seen in the analysis of Business Services over DOCSIS (BSOD) customers. This analysis does perform a CM-level evaluation, which will be presented in Section 3.2.2.

2.2. DAA Telemetry

One noteworthy enhancement of the DAA system is the improved real-time telemetry offered across each of the architectural components from the PPOD down to the CM. With the legacy system, telemetry data was only emitted at 5-minute intervals at the very least. With DAA, data is streamed with 15-second resolution. This improvement is key to delivering a speedy assessment of network health, as it eliminates the need to wait for post-deployment telemetry to become available. Additionally, this increased resolution into telemetry data allows the freedom to explore more check algorithms and provide higher quality network diagnostics. In *Solving The Mysteries of the Distributed Access Architecture*, Stehman et al. details the available telemetry across the access network and further expands on relevant topology (2021).

3. Methodology

3.1. Network Health Check Overview and Terminology

3.1.1. Pre-Check, Post-Check, and Metric Snapshots

The automated network health check is meant to detect and alert on any unintended customer impact after a software deployment. This of course entails checking live telemetry metrics directly after the deployment; however, to establish a PPOD-specific baseline on which to compare these post-deployment metrics, data collection and analysis is also needed prior to the deployment. As such, the health check was designed to be a two-part process, consisting of both a pre-check and a post-check. By design, a completed pre-check is required to start a post-check. From the PPOD CI/CD standpoint, this means that the API must be integrated both prior to and after the deployment.

Each of the pre- and post-checks can be further broken down into metric "snapshots". A metric snapshot can be defined as a collection of related telemetry metrics measured over the same time period and compiled to create a meaningful KPI. Since most of the evaluation algorithms function by comparing a given KPI at two different time points, each time-aligned metric snapshot can be considered a single comparison unit. Pre-check gathers metric snapshots from two different timeframes: *i.* over the course of history for the PPOD and *ii.* in the instant right before the deployment. Respectively, we will refer to these as "historical snapshots" ("historical-snaps" for short) and "pre-deployment snapshots" ("pre-snaps" for short). As historical-snaps often consider a wide timeframe prior to a deployment, typically an aggregation of telemetry data over time serves as the comparative metric value. This could be, for instance, the 10^{th} percentile of a PPOD's upstream packet rate over the past seven days.

The post-check works by querying and deriving metric snapshots from one timeframe only—in the instant right after the deployment. These snapshots will be referred to as "post-deployment snapshots" ("post-snaps" for short). The post-check also has the capability to load any of the cached historical-snaps or pre-snaps from the pre-check to perform the actual pre-to-post comparisons. Whether or not post-check borrows from the historical-snaps or pre-snaps depends on the KPI being compared. As a general rule of thumb, continuous KPIs, such as traffic flow, tend to be compared post-deployment-to-historical, whereas discrete KPIs, such as number of CMs online, tend to be compared post-deployment-to-pre-deployment. In adherence to DevOps daily maintenance schedules, the target completion time of a post-check, including all queries, data reads, comparison logic, and data writes, is two minutes or less. Figure 2 aims to further clarify the concepts of pre- and post-checks as well as metric snapshots.

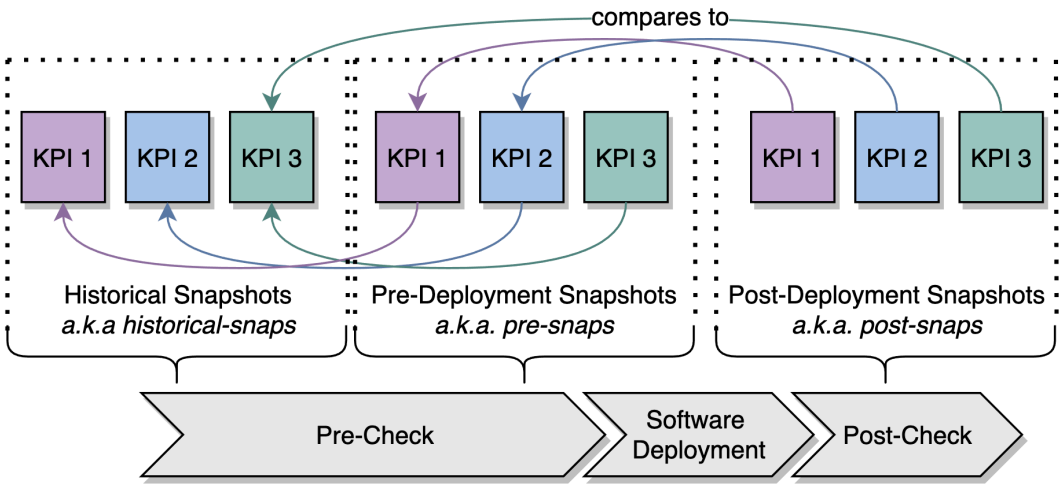

Figure 2: Conceptual relationship among pre-check, post-check, and historical/pre-deployment/post-deployment snapshots

It should be noted that the pre-check currently possesses the capability to compare pre-snaps to historical-snaps and assess network health prior to the deployment. The initial concept of the automated network health check was designed with this capability in mind so that a deployment could be automatically blocked if the network was deemed to be unfit for a software update. As the DAA initiative progressed, this feature was decommissioned, as there are other automated alert systems in place capable of blocking a deployment if the network is not considered healthy enough to undergo an update. Therefore, the main purpose of the pre-check currently is to gather the snapshots needed to perform comparisons in the post-check and to confirm that these snapshots are complete. In other words, the software deployment automation only interacts with network health assessments made in the post-check. Since the data sciences team has primarily focused on developing and optimizing the post-check functionality, this paper will focus on the assessment of KPIs in the post-check.

It should also be noted that the data sciences team's network health check is not the only post-deployment fail-safe in place within the deployment automation. The DAA engineering team has also incorporated a series of automated checks, more so pertaining to cloud microservice health; however, there can be slight overlap when it comes to the KPIs observed among the various checks. While these microservice checks are deemed to be another essential step within the deployment automation pipeline, they will remain outside the scope of this paper.

3.1.2. Service-Affecting vs. Non-Service-Affecting Updates

All DAA software updates, regardless of the component being updated, can be placed into one of two general categories: service-affecting (SA) or non-service-affecting (NSA). The distinction between the two depends on whether a component update will take customers offline for a brief period. SA updates, which are scheduled during maintenance windows (normally between 01:00 and 04:00 headend local time), typically involve RPD reboots causing all downstream customers to experience a brief expected service interruption. NSA updates, which comprise the vast majority of all DAA software updates, occur on components that often have service-preserving backup units, and will most likely not result in interruptions. In other words, SA updates can be considered high-risk, whereas NSA updates can be considered low-risk. The same automated network health check is performed on both SA and NSA deployments; however, it is important to make this distinction between the two categories when analyzing results, as SA updates tend to trigger more alarms than NSA updates, given that it takes some time for customers to come back online and the network to return to a normal, steady state following SA updates. As SA updates are high-risk and are purposely scheduled on just a handful of PPODs at a time, an operator typically oversees SA updates and interacts with the health check response live. Figure 3 shows examples of typical CM behavior over time during an SA update—specifically the steep drop-off in total number of CMs online during the RPD reboot, followed by a gradual recovery. The expectation for NSA updates is that a service interruption like this should *not* occur.

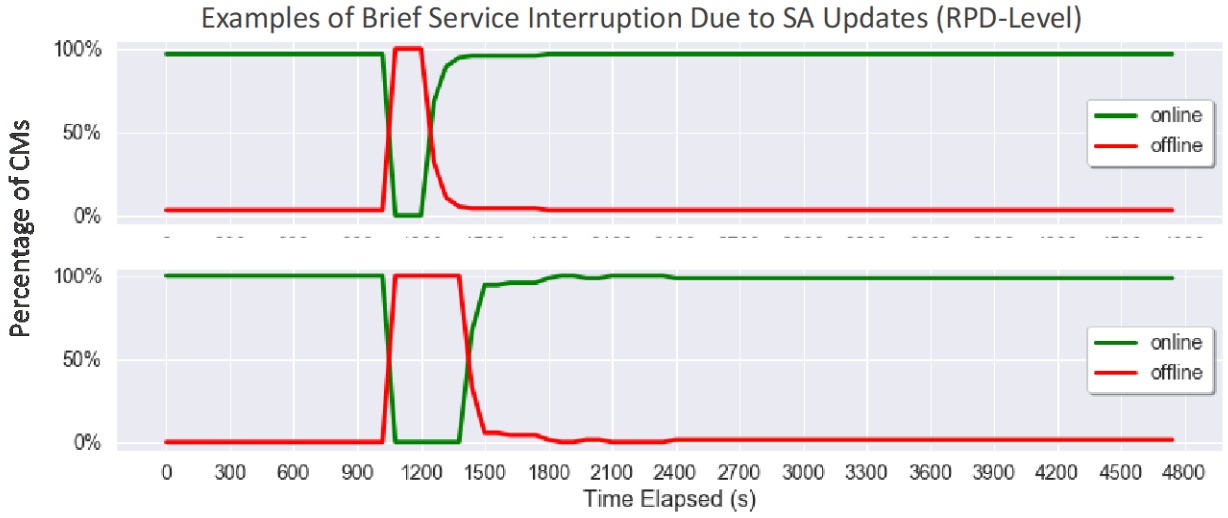

Figure 3: Example views of CM status vs. time during SA updates at the RPD-level

3.2. Assessment of Key Performance Indicators

The automated network health check, specifically the post-check, works by making rule-based comparisons between two snapshots of the same KPI. Thresholds and comparison algorithms are specific to each KPI and will be detailed in this section. When a particular KPI is evaluated against a threshold and breaks a rule, the KPI is assigned one of two categories: *warning* or *failure*. Whether or not a broken rule indicates a warning or failure depends on the KPI as well as the algorithm being used to assess it, both of which indicate the severity of the anomaly. Warnings can indicate signs of service degradation, but typically to a lesser extent than failures. As will be discussed in the next section, KPIs with warnings are intended to be observed and studied more closely but not quite considered severe enough to automatically alert an operator. The data sciences team has worked closely with DAA subject matter

experts (SMEs) in deciding which KPIs should belong in the failure category versus the warning category.

In addition to warnings and failures, there are a several KPIs which are calculated and presented in the API response for information only, as they are not indicative of service degradation nor are they held to a certain threshold. These KPIs are placed in a category called *info* and are meant to aid in observation and analytics.

3.2.1. Deciding When to Alert

The API response compiles all individual KPI results and provides a list of all KPIs which have failed the check and all KPIs which should deliver warnings. If a single KPI evaluation fails, the result of the entire health check is considered a failure, the PPOD is flagged, and an operator is alerted to take further action. KPIs in the warning category do not automatically fail the entire health check when a rule is broken. Rather, these KPIs are manually observed by DAA operators and SMEs as possible contenders for stricter treatment, further algorithm tuning, or even live operator intervention in the case of SA updates. It is not uncommon for KPIs to start out in the warning category and later be transferred over to the failure category when SMEs confirm the relevancy of the KPI, and an optimal algorithm has been decided. Figure 4 aims to conceptually demonstrate how the overall health check decision is calculated based on KPIs in the failures category *only*, despite KPIs in the warnings category.

Alongside the overall health check decision, a summary of the KPI readings and comparisons is presented to the operator as a diagnostic aid. For PPOD-level evaluations, a summary might include, for instance, CM counts for given states pre- and post- deployment and list any CMs which have changed state following a deployment. If an RPD-level evaluation fails, typically the summary will detail KPI readings and comparisons for the problematic RPD(s).

With this information, an operator can decide on a mitigative course of action in the event that service degradation is detected. For instance, an operator might perform an RPD reboot and retry the post-check to see if detected service degradation has been fixed. If nothing can be done remotely, an operator might decide to send a technician out into the field to intervene with RPD hardware. If a root cause cannot be identified, the operator might ultimately decide it is best to roll back the software update until further troubleshooting can be performed.

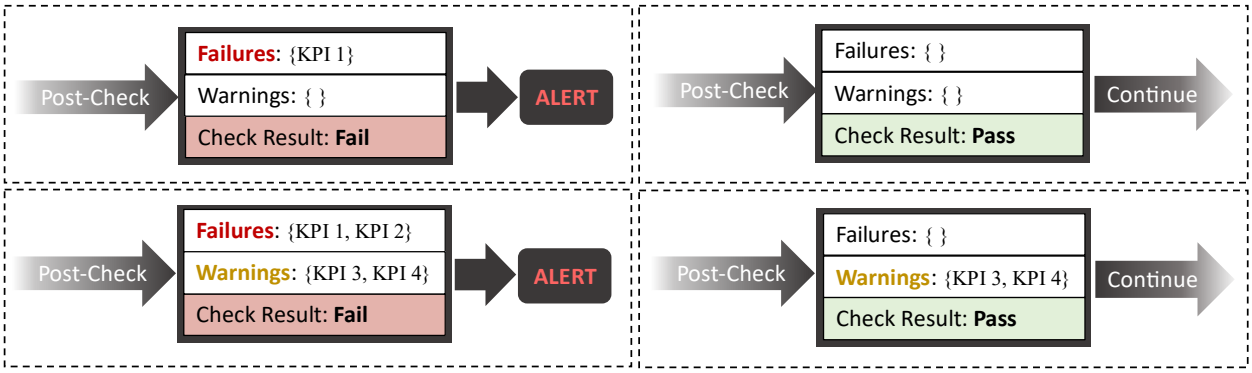

Figure 4: Some conceptual check outcomes

3.2.2. Algorithms and Thresholds

As discussed previously, most algorithms operate at either the PPOD- or RPD-level. This means that more granular CM and traffic metrics will be aggregated to provide a big picture view of the customer experience downstream of the PPOD and each RPD. This might involve, for instance, counting the CMs in a particular state downstream of an RPD, or summing all upstream traffic across all RPDs to calculate a PPOD-level traffic measure.

KPI comparison algorithms can be generally categorized as one of the following: *i.* percent recovery*, ii.* percent increase*, iii.* greater-than-zero*,* or *iv.* custom. Simply put, algorithms that analyze components in "good states", such as *online*, *connected, synced*, etc., tend to use percent recovery calculations, while algorithms that analyze components in "bad states", like *partial service*, tend to use percent increase calculations. Some KPIs are analyzed simply by checking if the post-deployment reading is greater-than-zero. This is typically done with continuous traffic KPIs since the algorithm is only limited to an instant sample of data post-deployment, eliminating the possibility of any trend analysis. Custom algorithms, as the name suggests, do not fall into these general categories and are evaluated using customized logic and thresholds. Table 1, Table 2, and Table 3 summarize KPI algorithms, thresholds, and alert categories at the PPOD-, RPD-, and CM-levels respectively. These KPIs are observed in accordance with DOCSIS and are detailed in the latest CableLabs DOCSIS Remote PHY specification and database of DOCSIS Management Information Bases (MIBs). It may be helpful to reference the "Abbreviations" section on page 24 to comprehend the KPIs presented.

As seen in these tables, several KPIs have two evaluation renditions. Typically, this is done when two levels of severity—one that results in a warning and one that results in a failure—are analyzed for the same KPI. Also noteworthy are the several algorithms/thresholds which have scenario-specific exceptions, as highlighted in the "Notes" section in each table. An example of this can be seen when CMs-in-partial-service is analyzed and held to a percent increase threshold (< *X* % increase), but the pre-snap count of partial CMs is very low (< 10). In a scenario like this, the addition of just a few more CMs in partial service post-deployment can cause an entire health check to fail, halting the deployment pipeline and alerting an operator. This scenario is likely not indicative of service degradation due to a software update given that it is not atypical to see some random partial service fluctuation. In this scenario, an exception would be programmed which would forgive a few additional CMs in partial service despite technically breaching the percent increase threshold.

Table 1: Summary of PPOD-Level KPI Algorithms and Thresholds

Level	KPI	Algorithm	Threshold	Notes	Category
PPOD	RPDs Online	percent recovery	> *X* % RPDs	test/pre-production RPDs omitted from calculation	failure
	CMs Online - Overall	percent recovery	> *X* % CMs	uses subset of CMs online in pre-snap	failure
	CMs Online - IP v4/v6	percent recovery	> *X* % CMs	IP version breakdown, uses subset of CMs online in pre-snap	warning
	CMs Online - BSOD	percent recovery	> *X* % CMs	-	failure
	CMs in Partial Service	percent increase	< *X* % increase in partial CMs	custom algorithm for low-CM scenarios	failure
	CPE Types Online	custom	N/A	breakdown by CPE type	info
	CPE Types in Partial Service	custom	N/A	breakdown by CPE type	info
	MD DS/US Traffic	greater-than-zero	packet rate > 0	except when packet rate is historically zero	warning
	Partial Service (Statistical Percentages)	custom	not statistically greater than history (< 3σ)	calculates historical distributions of percent-CMs-in-partial	warning
	RPDs PTP-Synced	percent recovery	> *X* % RPDs	-	failure
	RPD Time Offline	custom	N/A	time each RPD is down during SA event	info
	PCMM Connection 1	greater-than-zero	COPS connected/open > 0	-	failure
	PCMM Connection 2	percent recovery	> *X* % COPS connected/open	-	warning
	OFDMA Channels	custom	N/A	breakdown of OFDMA channels	info
	Mid-Split Enabled CMs	percent recovery	> *X* % mid-split enabled CMs	includes breakdown of mid-split enablement status pre- and post-deployment	warning
	Mid-Split Utilizing CMs	percent recovery	> *X* % mid-split utilizing CMs	includes breakdown of mid-split utilization status pre- and post-deployment	warning
	Mid-Split Enabled RPDs	custom	N/A	breakdown of mid-split enabled RPDs	info

Table 2: Summary of RPD-Level KPI Algorithms and Thresholds

Level	KPI	Algorithm	Threshold	Notes	Category
RPD	CMs Online - Overall 1	percent recovery	> *X* % CMs per RPD	uses subset of CMs online in pre-snap	warning
	CMs Online - Overall 2	greater-than-zero	> 0 CMs per RPD	except when RPD had zero CMs in pre-snap	failure
	CMs Online - IPv4/v6	percent recovery	> *X* % CMs per RPD	IP version breakdown, uses subset of CMs online in pre-snap	warning
	CMs Online - BSOD	percent recovery	> *X* % CMs per RPD	uses subset of CMs online in pre-snap	failure
	CMs in Partial Service	percent increase	< *X* % increase in partial CMs per RPD	custom algorithm for low-CM scenarios	warning
	CPE Types Online	custom	N/A	breakdown by CPE type	info
	CPE Types in Partial Service	custom	N/A	breakdown by CPE type	info
	MD DS/US Traffic	greater-than-zero	packet rate > 0 per RPD	except when packet rate is historically zero	warning
	DSG Traffic 1	greater-than-zero	packet rate > 0 per tunnel, per channel, per RPD	warns if a single tunnel has zero traffic post-deployment for a single RPD	warning
	DSG Traffic 2	greater-than-zero	packet rate > 0 per tunnel, per channel, per RPD	fails if all tunnels have zero traffic post-deployment for a single RPD	failure
	OFDMA Channels	custom	N/A	breakdown of OFDMA channels	info
	Mid-Split Enabled CMs	percent recovery	> *X* % mid-split enabled CMs per RPD	includes breakdown of mid-split enablement status pre- and post- deployment	warning
	Mid-Split Utilizing CMs	percent recovery	> *X* % mid-split utilizing CMs per RPD	includes breakdown of mid-split utilization status pre- and post- deployment	warning

Table 3: Summary of CM-Level KPI Algorithms and Thresholds

Level	KPI	Algorithm	Threshold	Notes	Category
CM	BSOD DS/US Traffic	greater-than-zero	packet rate > 0 per CM	except when CM historical traffic is also zero	warning

One noteworthy feature of our approach is the ease with which custom algorithms and more sophisticated models can be incorporated. Data required to implement a custom algorithm or model is simply gathered and processed, often in-query, to produce the desired data format as an abstract metric snapshot unit. This collection step often takes direction from a configuration file, which might specify the timeframe over which to gather data, for example. The same steps which perform comparisons and evaluate simple algorithms against thresholds can be easily abstracted out to incorporate models instead, all while delivering results and summaries in a format consistent with that of other evaluations. An example of a custom algorithm is discussed in the next section.

3.2.3. Custom Algorithm Example – Partial Service

One particular KPI worth discussing in-depth is partial service. Partial service occurs when a CM is online but unable to operate on one or more downstream (DS) or upstream (US) channels, which may or may not hinder the customer experience. A CM can go into partial service for a variety of reasons, including loss of communication on a channel, inability to acquire a channel, and/or configurational incompatibilities. Despite its negative connotation, partial service is actually a beneficial feature in that it can often allow an impaired CM to have a mostly normal transmit/receive experience on the subset of channels it has available (Volpe, 2011). Nevertheless, partial service is an indicator that a customer is either currently experiencing service degradation or at risk for service degradation in the future; therefore, partial service is an important KPI to monitor during a software update.

As indicated in the algorithm exception example discussed in Section 3.2.2, partial service can be a notoriously difficult metric to analyze pre-to-post deployment. The difficulty lies in the fact that: *i.* the analysis is limited to a brief sample of data instantly after the deployment, as long-term trend analysis is not in compliance with the two-minute check execution window *ii.* partial service can naturally fluctuate due to factors unrelated to DAA software updates, and *iii.* the expected effect of a particular software update on partial service is not always known. As demonstrated in Table 1 and Table 2, there are standard percent increase algorithms in place to analyze partial service. These algorithms vary in efficacy depending on the size of the CM population observed and tend to capture blatant partial service issues but might not adequately flag more subtle post-deployment partial service anomalies. It is not obvious how to define a more sensitive threshold using the percent increase strategy without introducing excessive false positives; therefore, a new-and-improved custom algorithm was developed. This novel partial service algorithm attempts to further capture PPOD-specific partial service anomalies using a statistical approach.

The steps to this approach can be summarized as follows: *i.* perform a historical query in the pre-check to get a sample of partial service snapshots, *ii.* form distributions of percent-CMs-in-partial specific to the PPOD over several time periods, *iii*. assume normality and calculate thresholds for each distribution by considering statistical convention “three standard deviations above the mean” (a.k.a. 3σ) to be the cutoff, *iv*. calculate instant post-deployment percent-CMs-in-partial and compare to the thresholds. Essentially, this algorithm answers the question: “Is partial service outside the normal range following a software update?”. Figure 5 aims to visually depict how these steps are used to evaluate the KPI. An instant pre-snap reference point is also included to indicate the difference pre-to-post deployment in relation to the historical distributions.

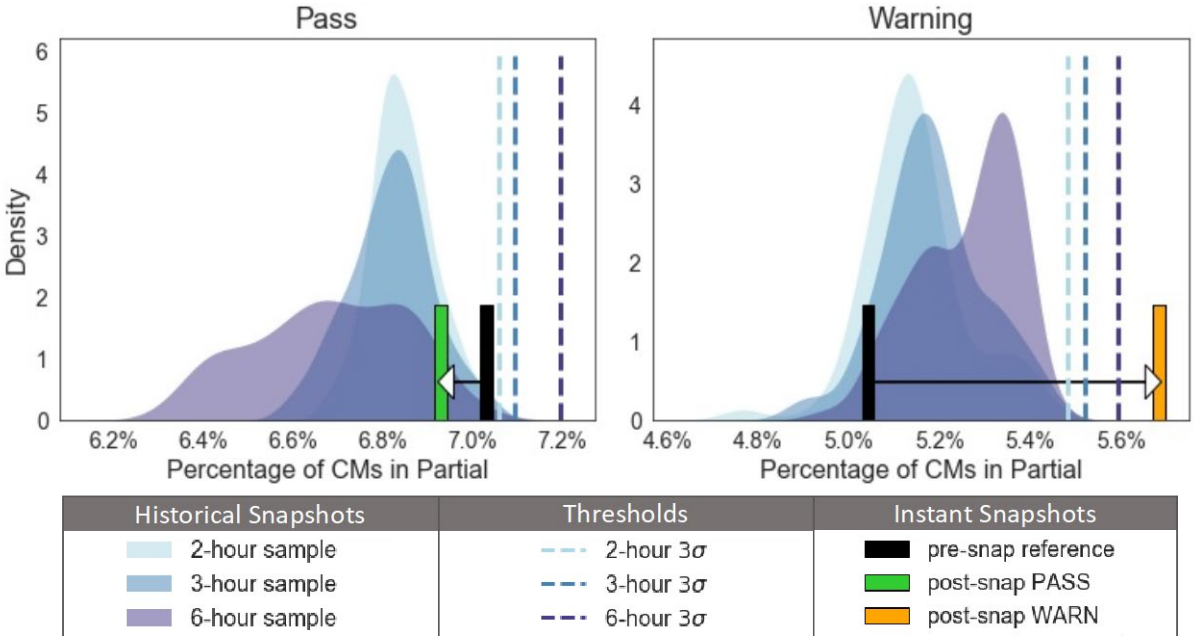

Figure 5: Visuals demonstrating statistical partial service algorithm

3.3. Integration with Software Deployment Automation

As previously mentioned, the network health check integrates with each CI/CD pipeline both prior to and after the software deployment for pre-check and post-check respectively. This setup technically requires four endpoints: *i.* initiation of pre-check, *ii.* polling of pre-check status, *iii.* initiation of post-check, and *iv.* polling of post-check status. The reason for this setup is that each check execution can take up to several minutes, which can exceed the connection time limitations of the cloud resources chosen to implement this application. Check executions are run as asynchronous background processes kicked off by the initiation steps, and the polling steps are intended to deliver a quick indication as to whether the check is complete or still running. Therefore, polling loops are needed within the automation to continue polling a check until the check is complete and delivers results.

When a pre-check is started for a PPOD, a universally unique identifier (UUID) is generated and passed back in the response of the first endpoint call. This UUID is used for reference throughout the remainder of the workflow to ensure each step accesses the correct cached snapshots and process metadata. UUIDs also serve the purpose of representing unique PPOD/software update/timestamp combinations, which is helpful in debugging and analytics.

The application is capable of post-check retries, which take new post-snaps and compare them to the same pre-/historical-snaps when the post-check initiation endpoint is called again. This feature can come in handy particularly during SA updates, when network recovery is gradual and variable from scenario-to-scenario. Typically, a retry is a manual process kicked off after an operator has waited for recovery following an SA update or intervened to correct a detected network issue. Figure 6 demonstrates how the health check features discussed above integrate to form a CI/CD workflow.

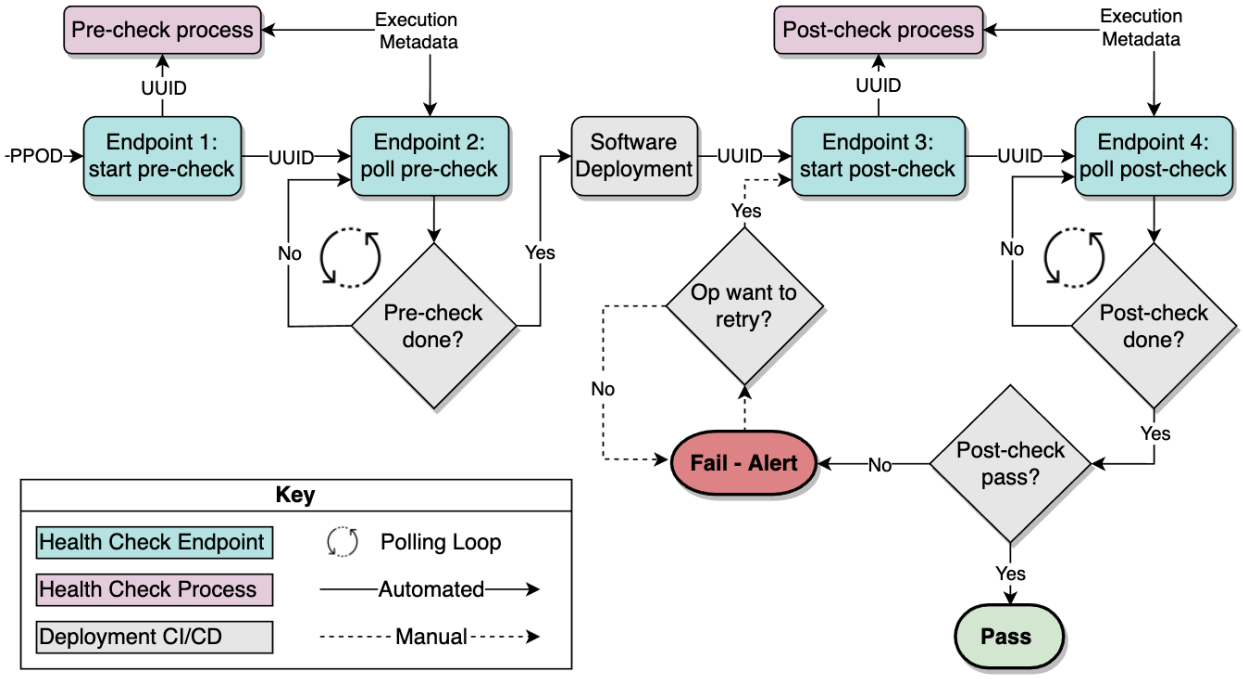

Figure 6: CI/CD integration with health check API — an example workflow

3.4. Health Check Cloud Environment

The health check operates entirely in a cloud environment. The API is hosted using a cloud API gateway service and interacts with a state machine, which orchestrates the health check workflow. A single state machine is used to manage both pre- and post-check, allowing each execution to be defined by the UUID described in the previous section. This is made possible by using callback functionality, which waits for a token to be passed back to the workflow before moving from pre-check to post-check (or from post-check *n* to post-check *n+1*). Check logic, which includes telemetry queries, comparisons, evaluations, etc., is carried out using serverless cloud workers. Check execution metadata, cached snapshots, and final results are all stored in a cloud database, which can be queried upon calling the API polling endpoints to retrieve the API response. The response should either indicate that a check is still running or deliver the completed check results. Unexpected errors (e.g., errors connecting to or querying from the telemetry database) would also be detailed in the response so that a retry can be performed. Verbose check data intended only for *ad hoc* analysis is also stored in the cloud database. Figure 7 presents a diagram of the health check cloud environment, demonstrating the workflow and interactions among cloud services.

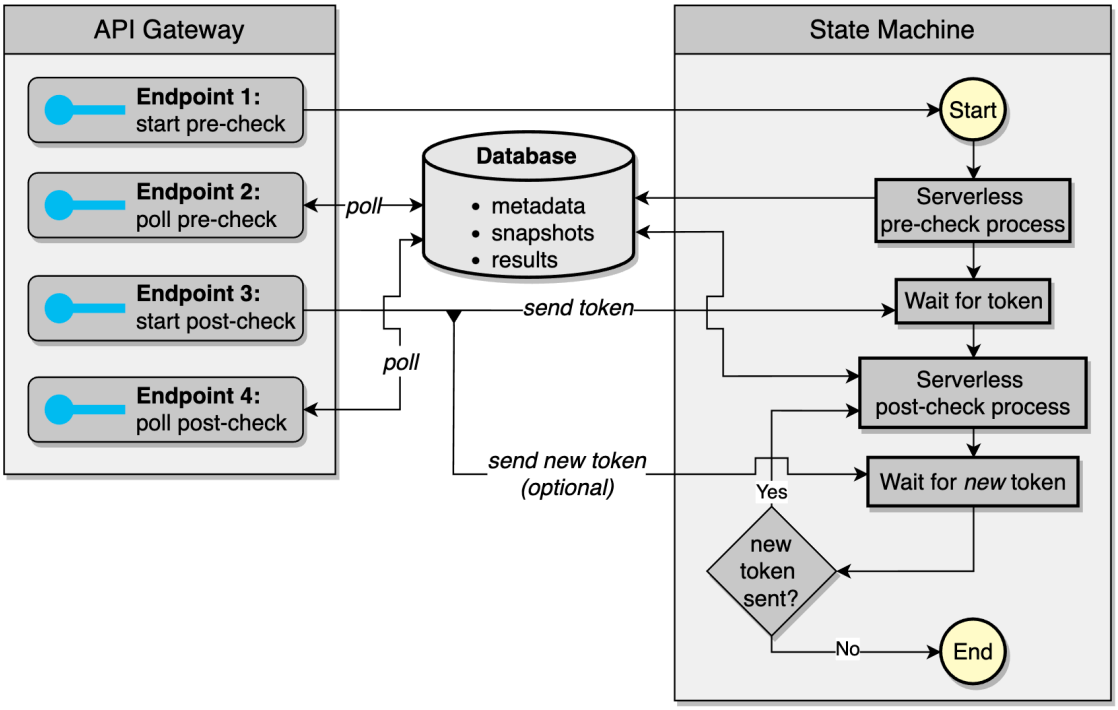

Figure 7: Diagram of cloud environment

4. Discussion

4.1. Usage Analysis

Since the very first production release in 2020, usage of the automated network health check has increased drastically and should continue to increase as Comcast scales DAA deployments. During the initial release stage, API calls were relatively sparse, as the DAA footprint at the time was much smaller, and the automation initiative was only beginning. Nowadays, it is not uncommon to see the API called hundreds, if not thousands, of times per day, depending on the components scheduled for maintenance. The health check is constantly interacting with more and more PPODs, customers, and types of software updates, as DAA continues to scale and engage in automation efforts. As demonstrated in Figure 8, this upward-trending call rate aligns proportionally to the standup of new digital clusters[1], which is a proxy to the growth rate of the DAA footprint.

While this substantial increase in call rate is undoubtedly a testament to the utility of the health check, it is also an indicator that the data sciences team must consistently perform due diligence to ensure that the check infrastructure and selected cloud resources continue to scale to meet the needs of the DAA automation initiative. For instance, cloud environment settings—namely microservice memory, provisioned concurrency settings, and programmed timeouts—are frequently tweaked to maintain reliability and performance of the health check. Additionally, telemetry queries performed in the health check are stress-tested frequently to ensure that the telemetry database can safely handle the large request loads typically seen during a burst of concurrent network health checks.

[1] The health check was released and integrated in 2020; however, health check data was not stored in an optimal format for analytics until later. Therefore, visuals will only show trends for more recent timeframes.

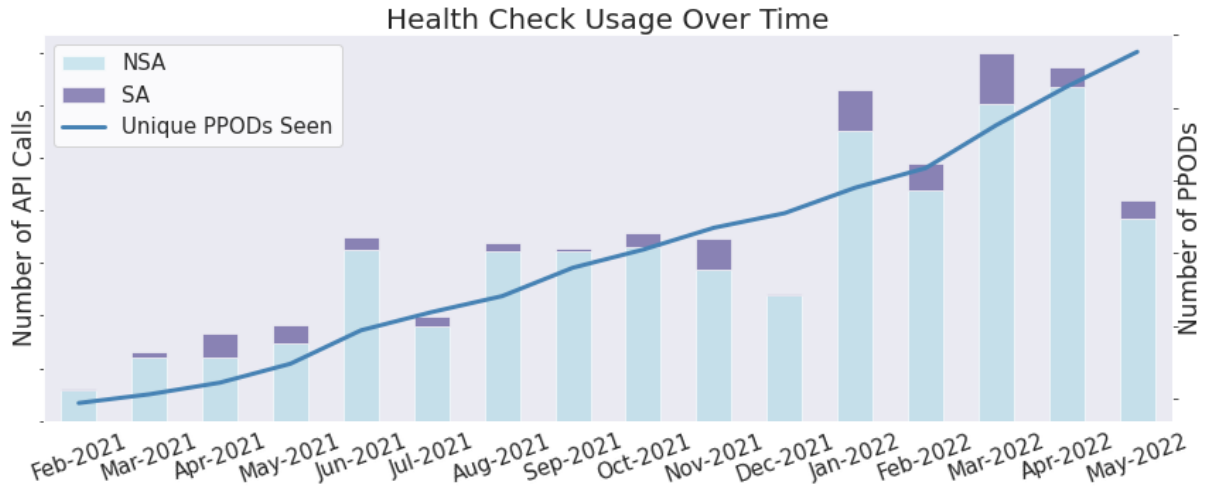

Figure 8: Health check usage

4.2. Analysis of Check Results

To depict failure and warning rates, Figure 9 breaks down the percentages of passes, failures, and warnings for both SA and NSA updates respectively over the past year[2]. As expected, low-risk NSA updates appear to deliver a consistently high pass rate, with a very small percentage of calls resulting in failures. As demonstrated by this data and confirmed by DAA operators and SMEs, the health check has been optimized to a steady state and performs proper due diligence surrounding NSA software updates. The check is capable of alerting the ops team *only* when necessary, but does not over-alert with false indicators of service degradation. Achieving optimal sensitivity is especially important for NSA updates, as they comprise ~91% of all DAA software updates.

As expected, SA updates typically show a higher failure rate, given that customer recovery following an RPD reboot can be a gradual process and the automation can prematurely run the post-check before recovery is complete. Additionally, high-risk SA updates tend to need more network intervention than do low-risk NSA updates, even after a recovery period is observed. While these frequent failures might seem burdensome, they are typically no hinderance to the automation process, given that an operator oversees SA updates as part of maintenance protocol. In these failure scenarios, the main value of the network health check is in the post-check retry capability, not necessarily the alert functionality. The retry capability allows the operator to intervene and repeatedly check all 36 network KPIs with the click of a button, until customer recovery is complete and service is fully restored. As previously mentioned, the operator may also choose to act on warnings during SA updates.

The warning rate for NSA updates consistently hovers around 20%, while SA updates show a more variable warning rate. In both NSA and SA updates, a slight uptick can be seen in recent months, which is attributed to the launch of Comcast's mid-split trials and relevant KPIs recently introduced to the check for testing. These warnings are not necessarily causes for concern, but rather demonstrate the troubleshooting and finetuning process when new KPIs are added. The results of newly added KPIs are often studied on a case-by-case basis, validated against telemetry dashboards, and discussed with DAA

[2] Prior to August 2021, *warnings* were inconsistently defined and consisted of general comments to the end user in addition to observed KPIs. Therefore, results prior to this date will be omitted from analysis for the sake of consistency.

SMEs in terms of optimal sensitivity. Using this iterative process, we might adjust thresholds, add programmed exceptions, or even break out a single KPI into two KPIs, as seen done in Table 1 and Table 2. In Section 5.1, we will take a deeper dive into the mid-split initiative and discuss next steps for mid-split-related checks.

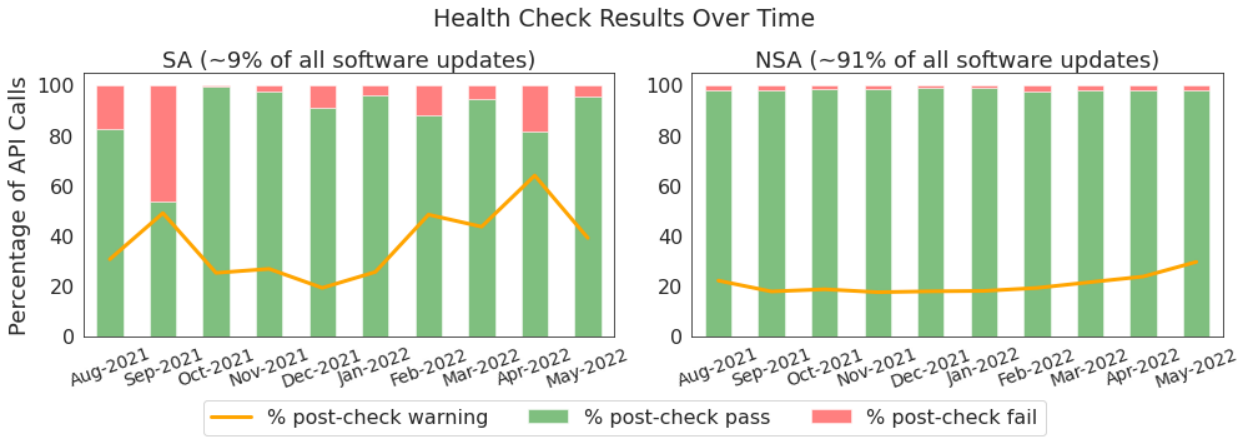

Figure 9: Health check results

We attempted a correlation analysis to better understand individual KPI result trends, but the findings are somewhat skewed, given that not all KPI results are independent of one another. Additionally, the large variety of component update types further confounds the analysis, but we will continue to seek out ways in which we can glean insight from the expansive data we have collected. We can, however, state that the two most common *independent* modes of check failure are PPOD-level/RPDs-online, which currently requires 100% RPD recovery following a software update, and PPOD-level/CMs-in-partial-service. These two failures have trickle-down effects on other KPIs; for instance, a missing RPD will also manifest as a DSG traffic failure and likely a CM-online failure, but the converses are not necessarily true. As warnings are constantly being tweaked and do not always have optimal thresholds, we will not discuss them in this context.

4.3. Lessons Learned

During the initial development stage of the health check, it was believed that only a few basic metrics—namely variations of RPD recovery, CM recovery, and CMs in partial service—would need to be observed. The initial design, which consisted of a handful of functions, was not well future-proofed, and quickly became unmanageable as DAA progressed and the need to observe new KPIs was realized. The team ended up refactoring the source code using an object-oriented approach to modularize the steps taken—namely metric snapshot querying, comparing snapshots, and evaluating against thresholds—so that new KPIs could easily be added without compromising the existing code. In retrospect, this strategy proved essential in optimizing the check to its current state, which evaluates more-than-triple the number of KPIs at inception. Figure 10 demonstrates the extent to which KPIs have been added over time.

Figure 10: Number of KPIs evaluated in health check over time

Another key lesson learned was "make everything configurable". Not only are algorithm thresholds configurable, but so are historical-snapshot timeframes and applied percentiles, the ability to assign KPI evaluations to the *warning*, *failure,* or *info* categories, the ability to turn on/turn off programmed algorithm exceptions, etc. This strategy greatly aided in the maintenance of unit testing and allowed finetuning of algorithms with a simple update to a configuration file.

Last but not least, optimizing storage of check data has greatly aided in our analytics, not only for informational purposes (like this paper), but also for debugging specific scenarios and further optimization of the health check based on retrospective analyses. As previously mentioned, we store all API responses, complete with health check results and summaries, in a database, and each is tied to a specific UUID for easy traceability. A verbose version of the check results, which contains more detail than the DevOps team requires but is helpful for our internal analytics and debugging, is also stored per each UUID. Additionally, placing check results in a relational database has vastly improved the efficiency of these analytics and has also enabled network health check results to be featured in other data science applications, such as Stehman et al.'s "Sherlock" analytics tool (2021).

5. Future Work

5.1. Continued Optimization of Nework Health Check: Mid-Split Updates

As alluded to previously, the data sciences team is in constant communication with DAA SMEs to discuss the addition of new KPIs and ways in which current KPI algorithms can be improved. This is expected to be an ongoing process as DAA progresses and Comcast launches other initiatives that overlap with DAA, particularly in support of 10G and full duplex (FDX) technology.

On the road to 10G and FDX, Comcast is actively working on deploying mid-splits to enable higher US and DS speeds within its digital footprint. Simply put, this is accomplished by incorporating an orthogonal frequency division multiple access (OFDMA) channel into the broadband spectrum, effectively doubling the available US spectrum (Olfert, n.d.). Enabling mid-split for customers across the DAA footprint requires a handful of configuration changes to each cluster's custom resource document (CRD). As such, the mid-split enablement process is orchestrated with the standard PPOD software update CI/CD automation, and the network health check is run after each attempted enablement.

To monitor the effectiveness and stability of each attempted mid-split enablement during the trial period, we started by adding several *info* and *warning* KPIs at both the PPOD- and RPD-levels, as it was not immediately clear what pass/fail criteria should be or if they were even needed. These KPIs looked at things like: CMs which successfully became enabled, CMs which became enabled but went into partial

service on the OFDMA channel, CMs which started seeing OFDMA traffic flow, OFDMA channels connected per RPD, etc. As referenced, in Section 4.2, we have seen a slight uptick in overall warnings rates due to this update, as we experiment and continue to tune these KPIs.

Observations like these have prompted further analysis and troubleshooting of the mid-split enablement process and have helped the data sciences team to brainstorm a few new purposeful rule-based algorithms. For instance, one KPI we intend to monitor more strictly in the future is CMs which go into partial service on the OFDMA channel due to mid-split enablement, as this is indicative that field technicians might need to intervene with an RPD's hardware to return network performance to a stable state. As the mid-split initiative expands, we will continually aim to identify meaningful KPIs and finetune algorithms, just as we have been doing throughout the lifetime of the network health check. The hope is that we can add KPIs and algorithms that will help operators to determine intervention strategies when mid-split specific service degradation occurs. Additionally, we will continue to perform analytics on mid-split results—*failure*, *warnings*, and *info*—to look for ways in which network monitoring and diagnostic reports can be improved.

5.2. Continous Monitoring

The network health check adds tremendous value to the DAA initiative in that it provides thorough network monitoring in the moments *right after* a software update and supports the goal of total automation by eliminating the need for manual observation during maintenance windows. This tool has been optimized to catch many signs of service degradation so that an operator can take quick mitigative action. However, some signs of software-related service degradation do not manifest until several hours, or even days after the deployment. Similarly, some impairments are not easily detected with a quick telemetry sample and require long-term trend analysis to detect. Given that the network health check is only intended to take a quick on-demand snapshot of the network health, it is not a good tool for continuous monitoring. Therefore, there is a need to develop a new tool, which will expand on the network health check approach and add the capability for long-term network health monitoring of each PPOD following a software update. This tool will differ from other production network monitoring tools in that it will try to pinpoint signs of service degradation specifically caused by or correlated with software updates, as opposed to all signs of service degradation in general. This distinction is key in delivering an optimized diagnostic tool designed specifically for DAA operations.

5.2.1. Implementation

With continuous monitoring, post-deployment anomalies will be detected using rule-based and machine learning (ML) algorithms, as well as time series analysis techniques, pushing the network monitoring initiative to the next level with artificial intelligence (AI). The proposed workflow for continuous monitoring is illustrated in Figure 11, which depicts an anomaly detection algorithm consistently observing the network for a set period after deployment. In a similar manner to the live health check, the DAA DevOps team will receive alerts when service degradation is detected or concerning anomalies are noticed. However, the workflow will differ slightly in that the continuous monitoring tool will run as a cron job and periodically push alerts to a publish/subscription (pub/sub) service available to the DAA ops team.

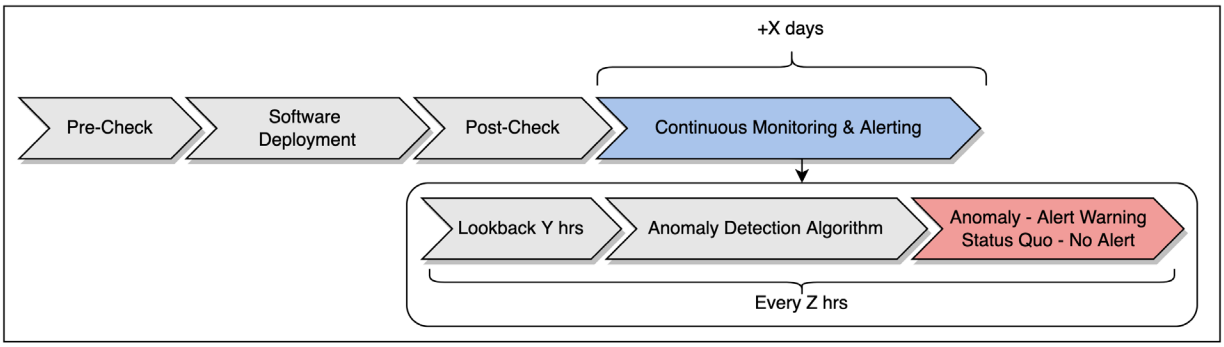

Figure 11: Continuous monitoring workflow

Continuous monitoring is considered an extension of the network health check but will be an independent product due to the magnitude and nature of monitoring metrics, algorithms, and alerts. Metrics in development for continuous monitoring include CM online status, partial service, MD traffic, customer contact metrics, and more. Figure 12 displays a recently noticed anomaly overlayed by a deep learning model in development to detect it. This particular anomaly is characterized by a select subset of CPE device types experiencing random brief service interruptions as a result of an erroneous configuration setting pushed through the deployment automation. This is an example of a scenario that went undetected with the instant telemetry analyses provided by the live network health check. An anomaly like this requires pattern analysis over time, making it an exemplary candidate for continuous monitoring.

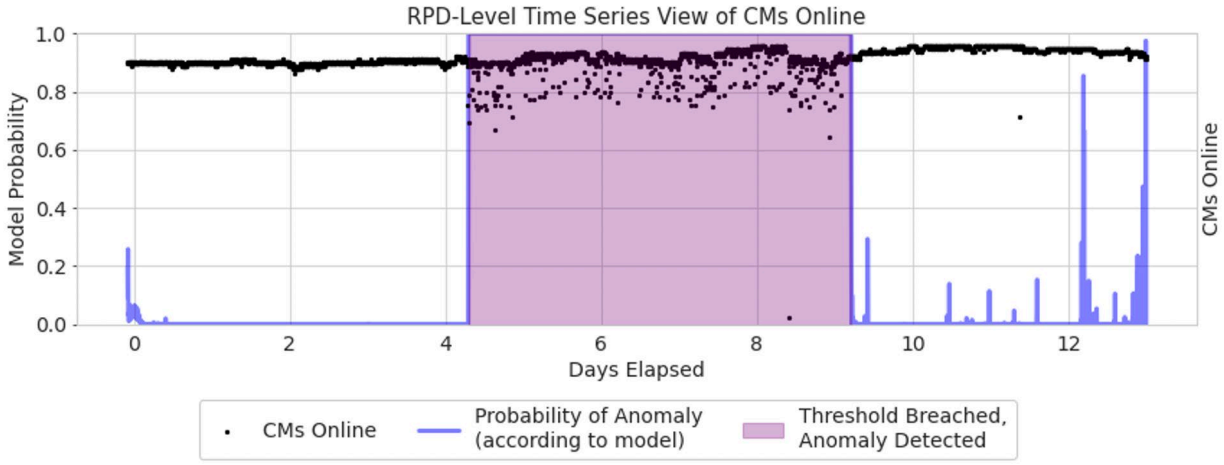

Figure 12: Proof-of-concept anomaly and model in development

5.2.2. Expectations

The data sciences team is currently testing the waters with some proof-of-concept models intended to capture anomalies following deployments in the long run. As these models improve and software-related anomalies are better understood, the hope is that continuous monitoring can play an even bigger role in the automation initiative—possibly even rolling back software automatically if it can be determined with near certainty that an erroneous deployment resulted in service degradation. With this initiative, we also hope to perform analytics at an even higher-level aggregation by examining groups of PPODs over the same timeframe following a common software update. Although we had mentioned that this would not be

a good approach for the live network health check, we hope that advanced time series techniques and possibly unsupervised methods, in conjunction with the expansive amount of data available, might be able to help identify problematic PPOD configurations, or even more specifically, erratic configuration/software interactions. In summary, the expectation for continuous monitoring is twofold: *i.* continue to bolster DAA automation by detecting and alerting on known anomalies and *ii.* use advanced analytics to increase our understanding of anomalous network patterns resulting from rapid scaling and constant enhancements occurring as part of the DAA initiative.

6. Conclusion

In response to the DAA engineering team's call to automate, the data sciences team has developed a live network health check meant to replace eyes-on-glass network health monitoring surrounding software updates. This was made possible thanks to the near real-time network telemetry streaming across the DAA footprint, allowing for quick and nimble analysis of network health KPIs. With the guidance of DAA SMEs and consistent feedback from the DevOps team, the data sciences team has been able to finetune and optimize the health check algorithms to achieve the dependable, steady decision engine currently in production today. While there have been a handful of key operational improvements that have supported the expansion of DAA, automated network health monitoring has been particularly impactful given the strict need to preserve the customer experience while performing updates and maintenance. With improvements like this health check, the footprint has been able to expand considerably, as made evident by the substantial increase in vCMTS clusters launched since the health check was first integrated. Despite the rapid growth of the DAA footprint, operational manpower needed to sustain the DAA initiative has mostly remained steady or even reduced in some scenarios, demonstrating the utility of the automated network health check.

While the health check is a valuable tool meant for use during brief maintenance windows, it is limited in that it cannot perform continuous monitoring over a more expansive timeframe following a software update. This is the next key need that the data sciences team will aim to tackle in support of DAA expansion and automation. With the continuous monitoring initiative, we intend to go beyond the scope of quick rule-based evaluations and enter the domain of anomaly detection and ML to deliver an even more thorough, diagnostic view of network health following software updates.

Abbreviations

API	application programming interface
AI	artificial intelligence
BSOD	business services over DOCSIS
CI/CD	continuous integration/continuous deployment
CM	cable modem
COPS	common open policy service
CPE	customer premises equipment
CRD	custom resource document
DAA	Distributed Access Architecture
DOCSIS	Data Over Cable Service Interface Specification
DS	downstream
DSG	DOCSIS set-top gateway
FDX	full duplex
HFC	hybrid fiber/coax
IP	internet protocol
KPI	key performance indicator
MD	MAC domain
MIB	management information base
ML	machine learning
NSA	non-service-affecting
OFDMA	orthogonal frequency division multiple access
PCMM	PacketCable MultiMedia
PPOD	physical point of deployment
PTP	precision time protocol
RPD	remote PHY device
SA	service-affecting
SME	subject matter expert
US	upstream
UUID	universally unique identifier
vCMTS	virtual cable modem termination system

Bibliography & References

Humanoids Optional: Deploying vCMTS at Scale with Automation, Bhanu Krishnamurthy and Gregory Medders, SCTE Expo 2021

Distributed Access Architecture Is Now Widely Distributed - And Delivering On It's Promise, Dr. Robert Howald et al., SCTE Expo 2021

Node Provisioning and Management in DAA, Robert Gaydos et al., SCTE Expo 2018

Solving The Mysteries of the Distributed Access Architecture, Matthew Stehman et al., SCTE Expo 2021

Data-Over-Cable Service Interface Specifications, DCA - MHAv2, Remote PHY Specification. CM-SP-R-PHY-I14-200323

http://mibs.cablelabs.com/MIBs/DOCSIS/. Accessed June 2, 2022.

Brady Volpe. "DOCSIS 3.0 Partial Service". The Volpe Firm, December 7th, 2011, https://volpefirm.com/docsis-3-0-partial-service/. Accessed June 15, 2022.

Matthew Olfert. "Getting Started with OFDMA". Broadband Library, n.d., https://broadbandlibrary.com/getting-started-with-ofdma/. Accessed June 15, 2022.

