Optimizing Spectrum Efficiency with Cloud-Based Load Balancing

A technical paper prepared for presentation at SCTE TechExpo24

Jay Zhu
Senior Principal Software Engineer
Comcast
Jay_Zhu@comcast.com

Defu Li
Distinguished Engineer
Comcast
Defu_Li@comcast.com

Andrey Skvirsky
Executive Director
Comcast
Andrey_Skvirsky@comcast.com

Qin Zang
Senior Manager
Comcast
Qin_Zang@comcast.com

David Wang
Software Engineer
Comcast
David_Wang@comcast.com

Belal Hamzeh
Vice President
Comcast
Belal_Hamzeh@comcast.com

Dan Rice
Vice President 2
Comcast
Daniel_Rice4@comcast.com

Mody Niv
Senior Vice President
Comcast
Mody_Niv@comcast.com

Table of Contents

List of Figures

List of Tables

Title **Page Number**

1. Introduction

The data over cable service interface specifications (DOCSIS®) specifications introduced channel bonding support since the 3.0 version [1], and continued through the 3.1 and 4.0 versions, which allows for scheduling of information in DOCSIS service flows over multiple channels concurrently and significantly increases cable modem (CM) usable capacity over DOCSIS 2.0. To define a set of logically bonded channels in either upstream or downstream, the definition of bonding group was created. A bonding group specifies references to a set of channels over which the cable modem termination system (CMTS) schedules data. In addition, the channel configuration, and the number of channels in the bonding group must be compatible with the bonding capabilities of the CMs being served.

As the CMs have variations in their bonding capabilities based on their DOCSIS technology versions and hardware, the definition and assignment of bonding groups are significant to ensure balanced sharing of the spectrum resource across all users and maximizing the available spectrum resources for CMs to reach their peak throughput. Because different CM bonding capabilities add variations to the maximum amount of spectrum resources CMs can utilize, balancing the load of all CMs within a service group can be pictured as stacking bricks with different widths and thicknesses, where the width represents the maximum single carrier-quadrature amplitude modulation (SC-QAM) and orthogonal frequency division multiplexing (OFDM) bonding capability of a CM, and the thickness represents the average utilization a CM uses on the channels it bonds to. With this analogy, the goal of channel load balancing becomes apparent – building a “flat wall” with a variety of bricks. Although, in the real-world, the scheduler can potentially help “water-fill” the traffic loads and allocate remaining channel capacities in a fine granularity.

Existing solutions today include static and dynamic load balancing. These solutions have been implemented and operated on different CMTS platforms over the years. The static load balancer specifies subgroups of CMs which may share the same channel sets, and it performs bonding group assignments at CM registration time. This improves resource sharing across the available spectrum on a CM-count basis. However, different users may have different service tiers, usage levels, and behaviors. Although the static load balancer is simple and robust, it does not account for real-time utilization at per channel level and per user level to make the optimal decision, which results in non-optimal spectrum resource efficiency.

In contrast, the built-in dynamic load balancer of a CMTS changes CMs’ bonding groups to dynamically maximize resource efficiency based on real-time CM counts or utilization. To operate the dynamic load balancer effectively, accessibility to sophisticated data such as CM make/model/device-class, policy configurations, and known CM bugs is required. However, the built-in dynamic load balancer has challenges and limitations to access such rich data, resulting in risks in production of encountering device bugs and customer service impacts. In the real-world, dynamic load balancing is often disabled because of such risks. With this many variables, it has been found that virtualizing the CMTS along with associated micro-services to apply the associated policy provides greater flexibility and simplicity than managing this many variables within an integrated cable modem termination system (I-CMTS).

In this paper, we introduce a cloud-based, external load balancer to solve the problems observed in both static load balancer and the built-in dynamic load balancer, such that the load balancer produces the optimal results by gaining access to key operation data while being robust, reliable, and low-cost. This is made possible by the virtualized cable termination system (vCMTS) application programming interfaces (APIs) and rich telemetry for service groups and CMs. In addition, we discuss architecturally how this functionality is separated from the vCMTS, reducing the development and release cycles of load balancing features and bug fixes, while being able to execute at flexible schedules and at scale. Finally, the experiment results are discussed.

2. Context, Observations and Imbalance Detection

In today's legacy (pre-DOCSIS 4.0) DOCSIS spectrum plan, there are less than or equal to 2 OFDM channels on the downstream and the D3.1 CMs can be bonded to all configured OFDM channels. On the upstream, the spectrum configuration allows for all non-DOCSIS 2.0 CMs to fully bond to all allocated SC-QAM channels and orthogonal frequency division multiple access (OFDMA) channels. Since the downstream and upstream schedulers can "water-fill" and balance the bandwidth allocations to CMs' bonded channels in real-time, the external load balancer today is designed to focus on ensuring fair-sharing of capacity among the downstream SC-QAM channels particularly for CM versions newer than DOCSIS 2.0. In addition, it is understood that all load balancing decisions are predictive – observations from the past are used to make bonding group assignment decisions. Having access to rich, multi-day historical data allows the external balancer to recognize the demand of rebalancing for each service group over an extended period and minimize the frequency of changes in the network.

To identify the opportunities for the external load balancer, the first step is to observe and analyze the existing imbalanced loads and their causes. A desktop application was developed to experiment and refine our imbalanced load detection algorithm with a highly interactive and high-performance graphical user interface (GUI) called "Utilization Playground." In the following sections, screenshots from this application are used to support illustration of our analysis and algorithms. The next section provides illustrative examples of different network dynamics that can result in less effective load balancing that can be improved by the external load balancing micro-service and should not be considered the typical state of the network. Improving these cases is still an important network function for optimal customer experience and capacity utilization across the network holistically.

2.1. Utilization Analysis and Case Studies

DOCSIS per-channel utilization is available at 15-second intervals across all service groups and channel types. This information is invaluable for researching and understanding capacity consumption behaviors. In a well-balanced service group, the utilization is uniformly distributed across all downstream SC-QAM channels. Periodic and daily demand changes appear in the time series graph as shown in Figure 1.

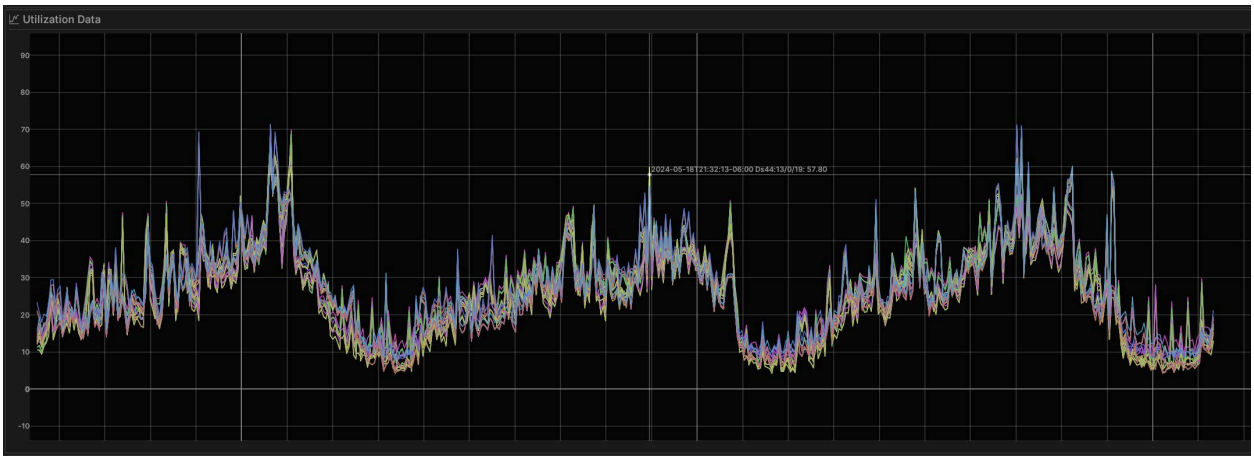

Figure 1 – Baseline: Balanced Utilization Distribution Across 44 DS SC-QAM Channels Over 3 Days

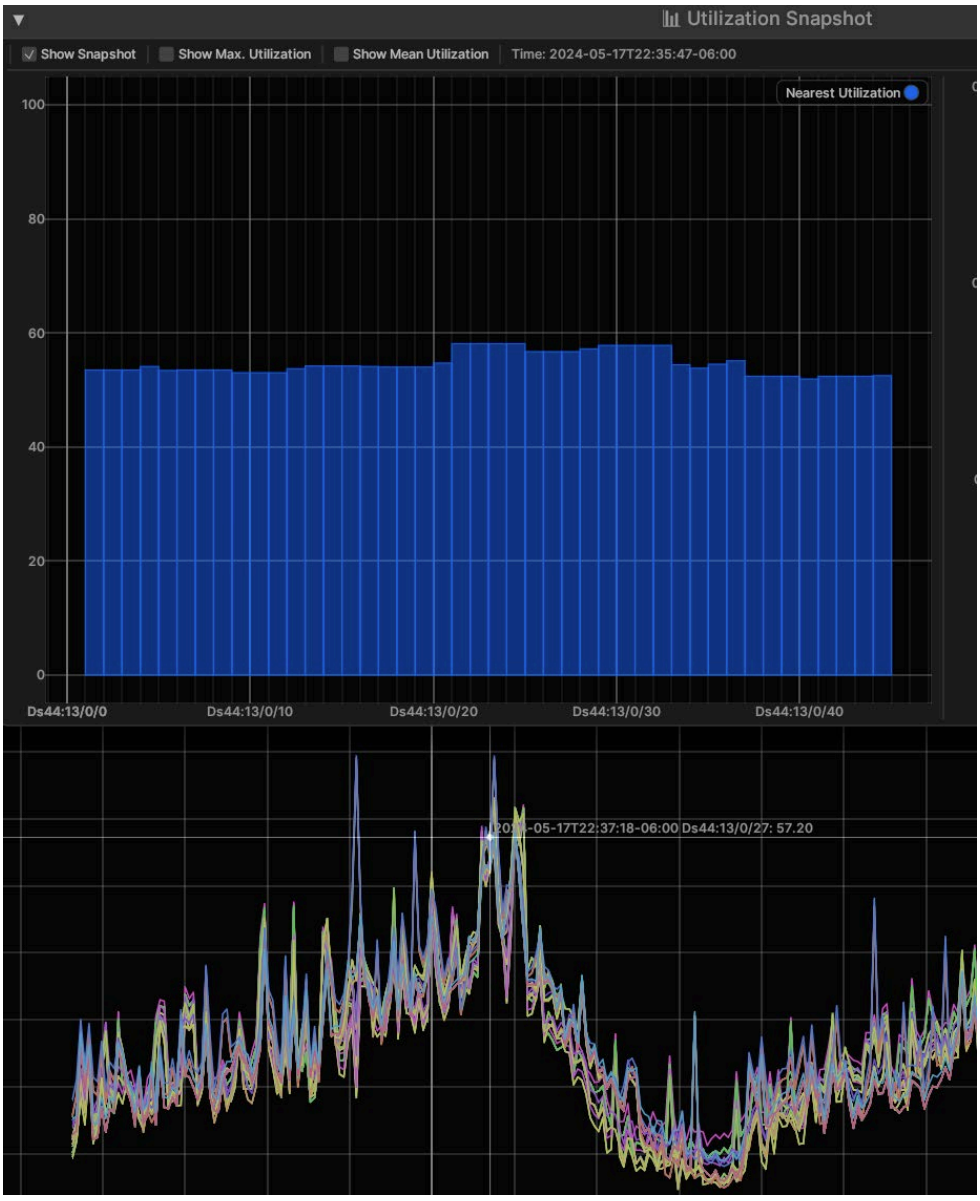

Figure 2 – Baseline: Utilization Distribution Snapshot Across 44 DS SC-QAM Channels During Peak Hours

In the utilization over-time graph in Figure 1, the x-axis represents the time, and the y-axis represents the utilization percentage at a given data capturing moment. For efficiency, the utilization sample interval was increased from 15 seconds to 300 seconds, which preserves most of the information needed by the analysis while significantly reducing the data input/output (I/O) and processing loads.

By inspecting further into the snapshots of utilization distributions provided by the "Utilization Playground" as shown in Figure 2, it is observed that the utilization is balanced across all downstream SC-QAM channels (bars from left to right in the bar chart) during the peak hours where imbalanced loads can cause the most impact to the customer experience. Both Figure 1 and Figure 2 demonstrate the characteristics of a well-balanced service group.

In comparison, a service group that is experiencing downstream SC-QAM channel utilization imbalance show prominently higher utilization across certain channel sets compared to other channels as shown in Figure 3 and Figure 4.

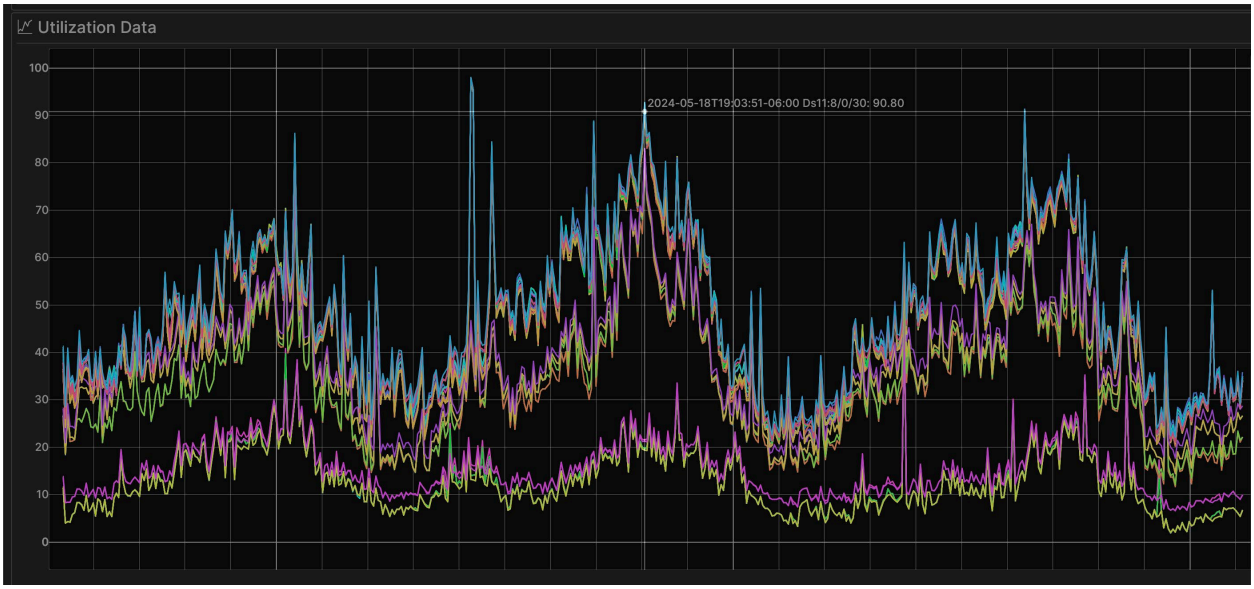

Figure 3 – Imbalanced Utilization Distribution Across 44 DS SC-QAM Channels Over 3 Days

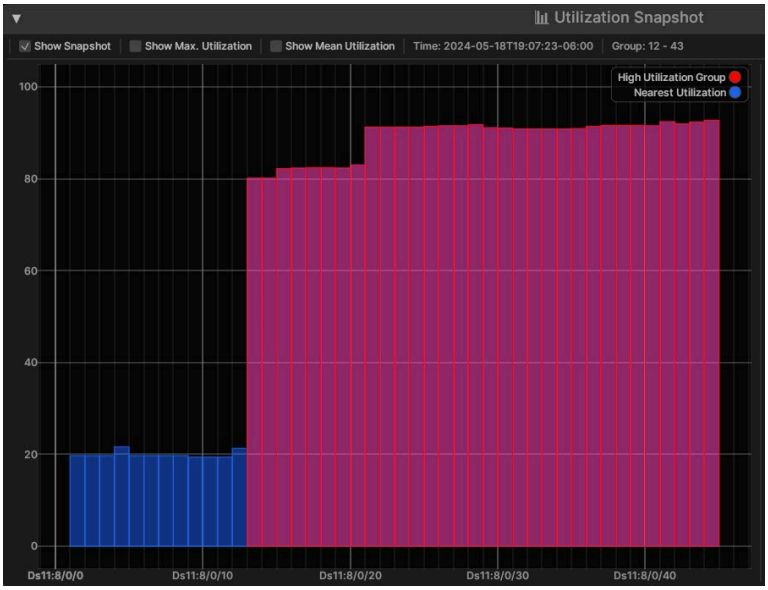

Figure 4 – Imbalanced Utilization Distribution Snapshot Across 44 DS SC-QAM Channels During Peak Hours

Upon inspection, the 24-channel and 32-channel bonding groups located at the upper end of the downstream SC-QAM spectrum appear as the cause of the imbalanced utilization. This assumption is further supported by the distribution of CM counts across bonding groups as shown in Figure 5 and Figure 6 before we consider possible uneven and outstanding bandwidth demands from individual CMs.

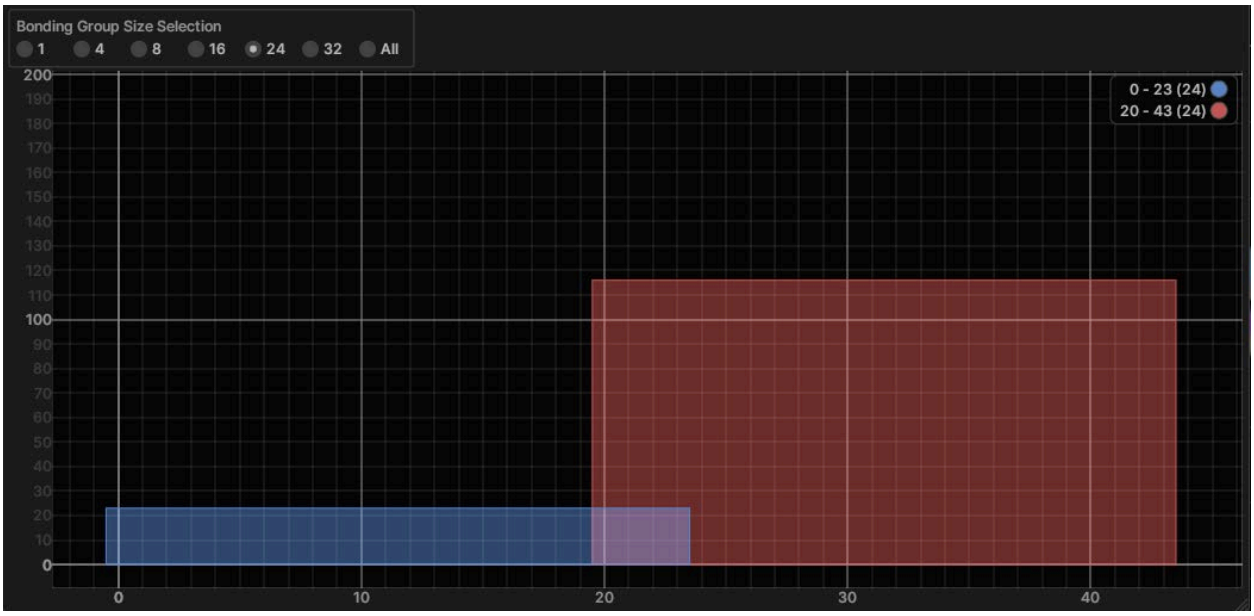

Figure 5 – Uneven CM Distribution Across 24-Channel Bonding Groups

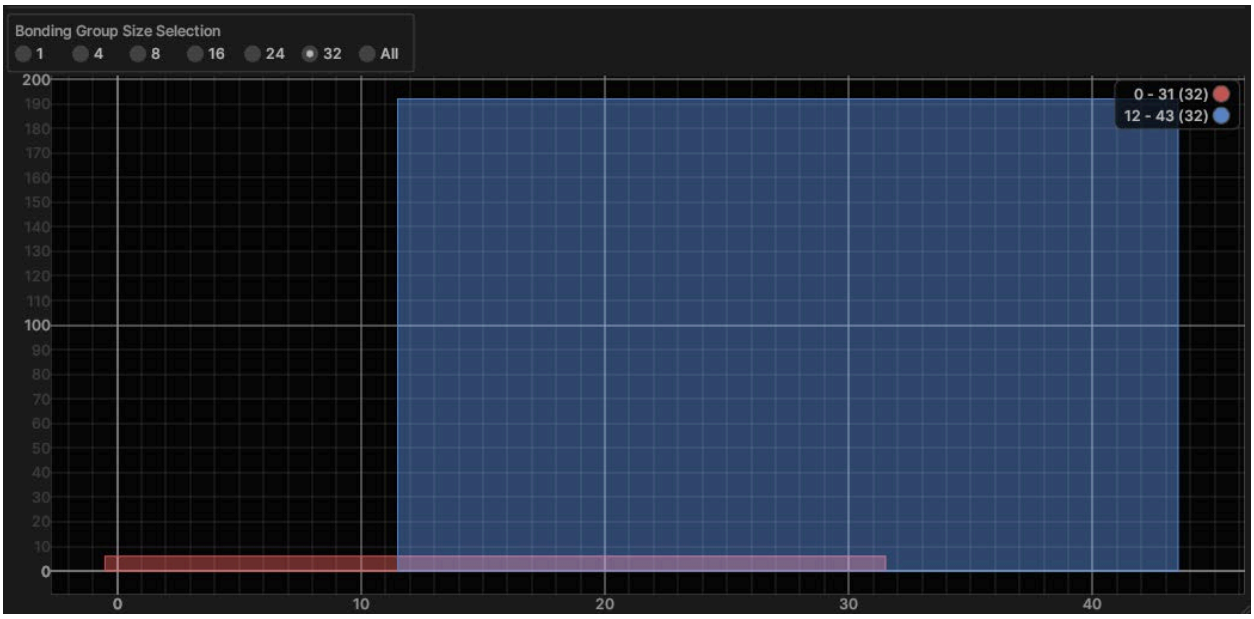

Figure 6 – Uneven CM Distribution Across 32-Channel Bonding Groups

The charts in Figure 5 represent the device counts of two 24 SC-QAM channel bonding groups, and the charts in Figure 6 represent the aggregated devices counts of two 32 SC-QAM channel bonding groups and two 32 SC-QAM plus one OFDM channel bonding groups. In Figure 5, the lower end bonding group is assigned to 22 CMs, whereas the upper end bonding group is assigned to 117 CMs. And in Figure 6, the lower end 32 SC-QAM channels are assigned to 6 CMs, whereas the upper end 32 SC-QAM channels are assigned to 191 CMs. This is a case where the static load balancer is having challenges to balance the load by distributing CM counts evenly across the spectrum by their maximum bonding capabilities.

Another example presented by Figure 7 and Figure 8, shows that a 16-channel bonding group at the upper end of the SC-QAM spectrum is highly utilized and causes imbalanced load. However, Figure 9 indicates

that all 16-channel CMs are allocated evenly across two 16-channel bonding groups. This observation suggests that different customer demands cause imbalanced loads as we would expect, but the static load balancer is unable to provide improvements as it does not consider per-CM utilization and usage behaviors.

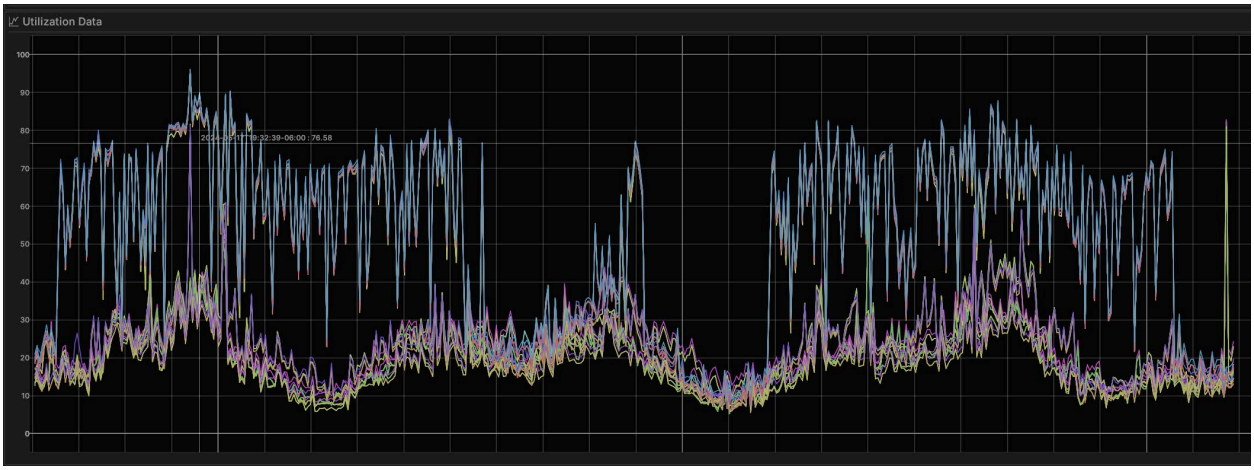

Figure 7 – Utilization Imbalance Caused by Uneven Bandwidth Demands Over 3 Days

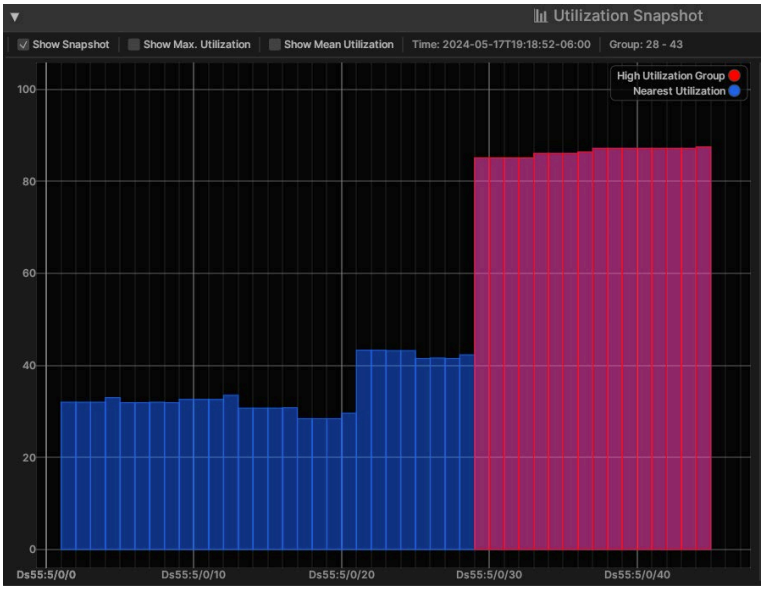

Figure 8 – Snapshot of Utilization Imbalance Caused by Uneven Bandwidth Demands

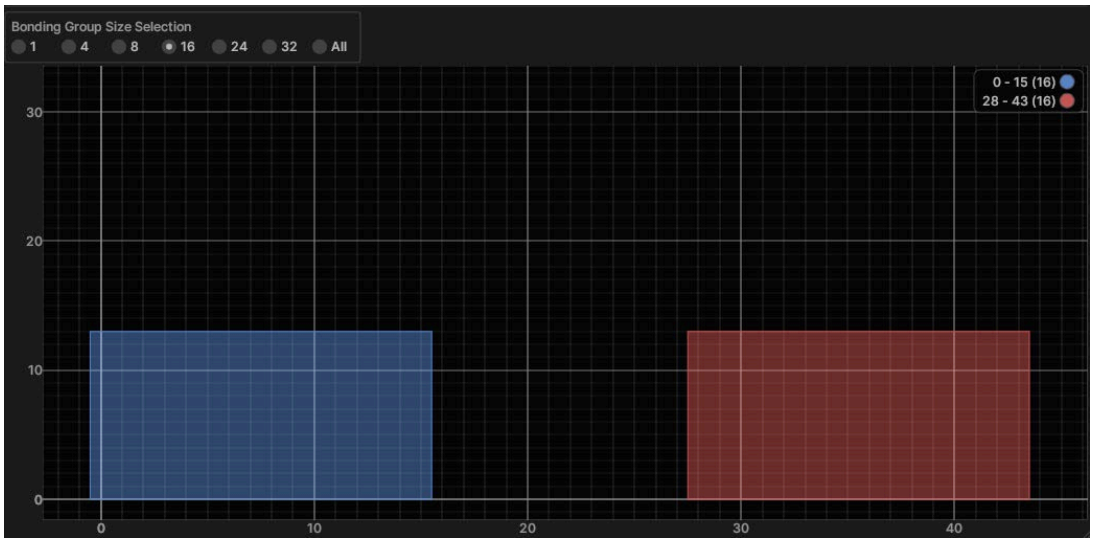

Figure 9 – Utilization Imbalance Not Caused by Uneven CM Count Distribution

In addition to seeing bonding groups showing imbalanced loads, the downstream primary SC-QAM channels can be saturated under certain conditions, such as:

- Devices are staying in partial service due to other issues.
- Abnormal DOCSIS 2.0 or set-top box devices.

A detected example of a highly utilized primary channel is shown in Figure 10 and Figure 11. In such cases, the load balancer is unable to improve the imbalanced loads. However, it is valuable to be able to identify, alert, and track such events for targeting and isolating primary channel utilization issues and suggesting improvements such as recommending customer device upgrades.

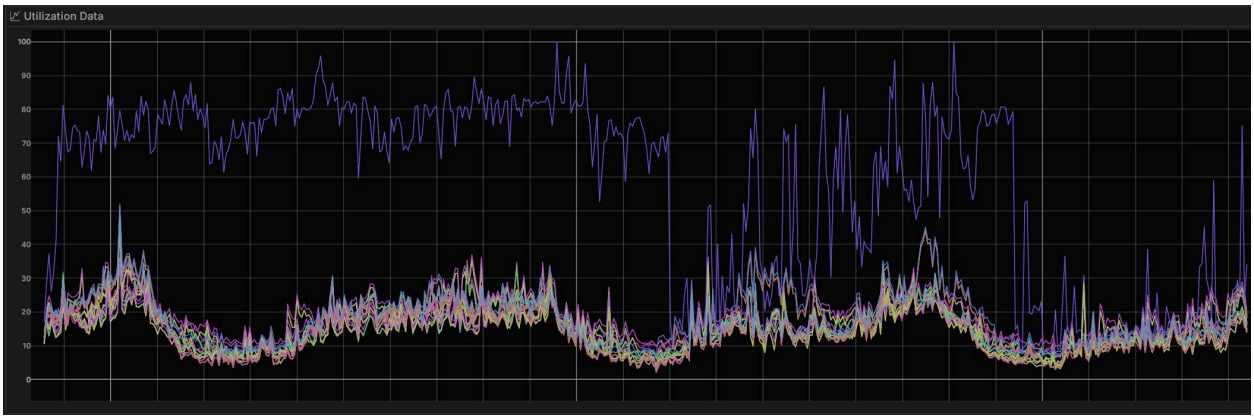

Figure 10 – Highly Utilized DS SC-QAM Primary Channel

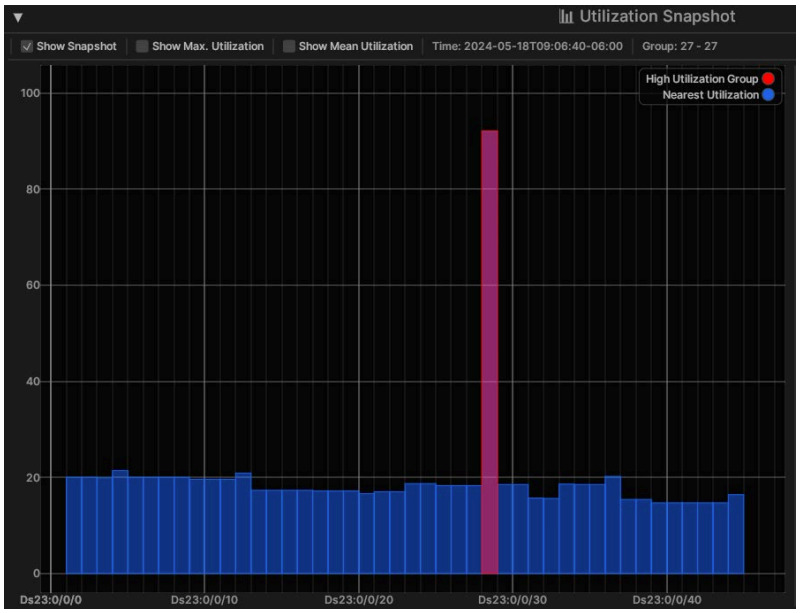

Figure 11 – Utilization Snapshot of Saturated DS SC-QAM Primary Channel

2.2. Imbalance Detection

With an understanding of the characteristics of the per-channel utilization and the causes of imbalanced loads, the detection process is automated by algorithms and services that audit all service groups at full production scale. This step is crucial for effectively selecting the target service groups for load balancing, reducing the overall cost of the external load balancer, and calculating load balancing key performance indicators (KPIs) for monitoring.

This imbalance detection algorithm is developed as part of a software library to be used by the extract, transform, and load (ETL) service. The primary requirements of this algorithm are:

- Detecting the imbalanced loads among utilization snapshots across the channels.
- Summarizing the percentage of time a certain service group is imbalanced.
- Identifying the top channel set that caused the imbalance within the utilization data duration and computing the distribution of percentages of time each channel set contributed to the imbalance.
- Calculating KPI's related to channel utilization and load balancing.

2.2.1. Algorithm

The first step of this algorithm is calculating the fairness measure across a snapshot of downstream SC-QAM channels' utilization values. The fairness measure employed is Jain's fairness index [2], it is defined as:

$$\frac{1}{1+\widehat{c_v}^2} \quad (1),$$

where $\widehat{c_v}$ is the sample coefficient of variation.

The Jain's fairness index effectively contains the fairness value to $\left[\frac{1}{n}, 1\right]$, or $\left[\frac{100}{n}, 100\right]$ if we multiply its value by 100 as visualized in Figure 12, where n is the number of downstream SC-QAM channels in our use case. This helps us reason and define the thresholds for the fairness measure.

Figure 12 – Jain's Fairness Index Over Time for the Multivariate Utilization Data

To detect imbalanced loads, the algorithm increments through the following steps:

1. Selecting time windows with significant overall utilization as S_1. An example is visualized in Figure 13.
    - When the overall utilization is low, the impact from imbalanced loads is small, and the fairness measure is not meaningful.
    - Selected time windows have their maximum (or mean) utilizations across all interested channels above the specified threshold such as 35%.
2. Selecting the time windows within the multivariate utilization time series which have their Jain's fairness index values less than or equal to a specified threshold such as 90%. These time windows are labeled as S_2. An example is visualized in Figure 14.
3. Computing the intersection time windows of S_1 and S_2 as S_3. The S_3 contains the region of interest of the imbalance observations on a service group. An example is visualized in Figure 15.
4. Filtering the service groups by the total identified imbalance time suggested by the S_3.
    - For example, the threshold may specify that the total time of S_3 must be greater than or equal to 30%, where the total time is the sum of all detected time windows in S_3.

This algorithm is designed to consume arbitrary duration of the multivariate utilization over-time data, but a 3-day or a 7-day data duration are commonly in use for considering non-temporary, sustained imbalance events.

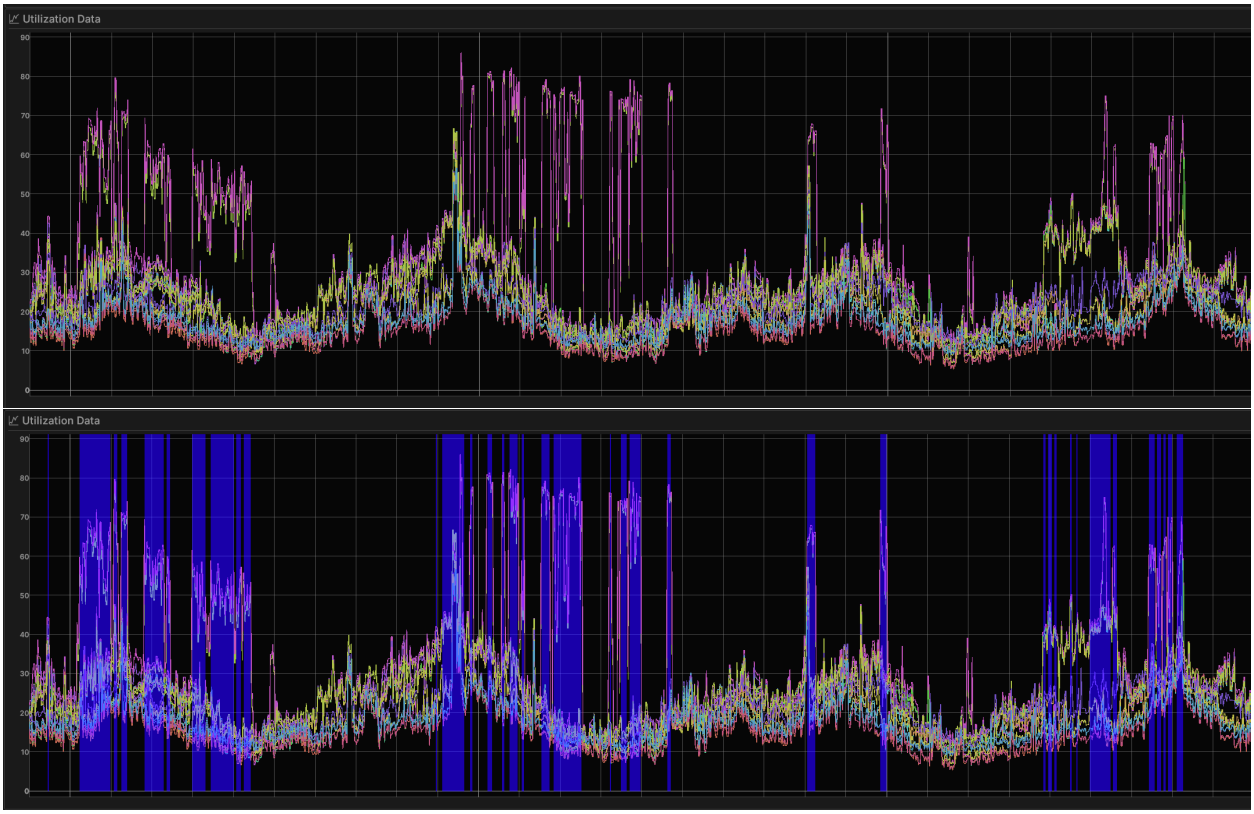

Figure 13 – Multivariate Utilization Data and Time Windows with Significant Utilization (Max. >= 40%)

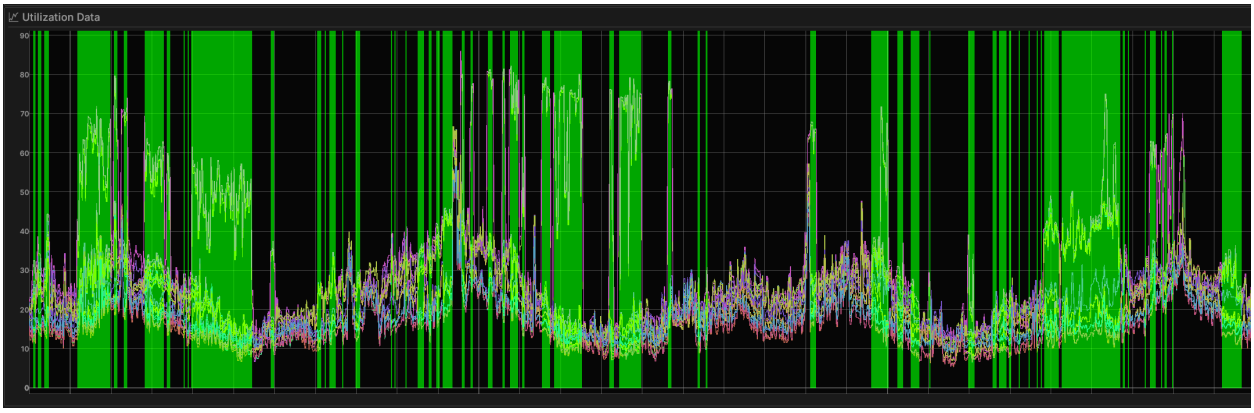

Figure 14 – Time Windows with Low Jain's Fairness Index (Fairness <= 95%)

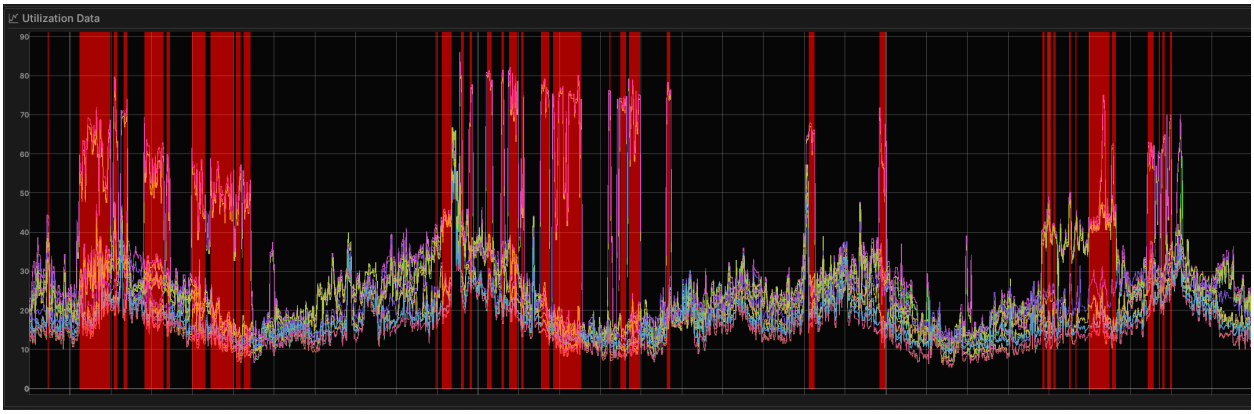

Figure 15 – Detected Imbalance Time Windows

The detection results suggest two categories of imbalance types and potential causes:

- Exceptionally high and lasting utilization on small bonding groups or primary channels.
  - This is often caused by users with sustained high demands (top-talkers) while using CMs with limited bonding capabilities.
  - This is sometimes caused by CMs with high bonding capabilities and high provisioning rates staying in partial service mode (caused by other issues).
- Moderate utilization imbalance caused by the same channel set(s) over a long duration.
  - This is often caused by limitations in the static load balancer where the CMs are not evenly distributed across the bonding groups and/or different user activity levels or provisioning rates are not considered.

To identify the above two types, for instance, the algorithm's thresholds are configured as:

- Abnormal utilization imbalance identification:
  - High utilization threshold for step 1 in the detection algorithm, for example, 80%, to target saturated small bonding groups and/or primary channels.
  - Low fairness index threshold for step 2 in the detection algorithm, for example, 80%, to target extremely imbalanced utilization patterns.
  - Low - moderate total imbalance time threshold, for example, 15% of the utilization data duration.
- Other (common) utilization imbalance identification:
  - Moderate utilization threshold for step 1 in the detection algorithm, for example, 30%.
  - High fairness index threshold for step 2 in the detection algorithm, for example, 95%.
  - High total imbalance time threshold, for example, 35% of the utilization data duration, to target sustained imbalance detection.

2.3. Top Channel Set Detection

In addition to the imbalance detection algorithm, we designed an algorithm to identify the channel set that caused the utilization imbalance, and we call this channel set the "top channel set." In the "Utilization Playground" application, this "top channel set" is automatically identified and highlighted in red color as shown in Figure 4, Figure 8, and Figure 11.

2.3.1. Algorithm

Identifying the top channel sets that caused the imbalanced utilization does not require per-CM usage data, it is done by executing the following steps:

1. For each utilization snapshot of all SC-QAM channels, use kernel density estimation (KDE) to calculate discrete distribution values from the estimated probability density function (PDF).
2. Use peak finding to identify the top utilization channel set in each utilization snapshot of all channels.
3. Find the channel set which appears as the top channel set in the greatest number of utilization snapshots.

The KDE is defined as:

$$\widehat{f_h}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \quad (2),$$

where n is the number of observations, K is the kernel – a non-negative function, and h is the bandwidth – a positive number that defines smoothness of the density plot.

For the bandwidth, Scott's rule [3] is used. It is defined as:

$$h \approx 1.06 \cdot \widehat{\sigma} n^{-1/5} \quad (3),$$

Finally, Epanechnikov kernel is used:

$$\mathrm{K(u)} = \frac{3}{4}(1 - \mathrm{u}^2) \cdot 1_{\{|\mathrm{u}|<1\}} \quad (4),$$

where u is the standardized distance between a data point and its estimation point and $|u| < 1$ defines the support where $K(u) = 0$ when the condition is false.

In summary, this algorithm uses KDE, PDF, and peak finding to perform 1-D data grouping/clustering at extremely high performance. Within each utilization snapshot, the channels are associated with the groups that they belong to by identifying their closest peaks found in the discrete distribution values from the PDF. This is visualized in the top-right corner of Figure 16 along with the utilization over-time data and a utilization snapshot of 44 downstream SC-QAM channels at the hovered time in the "Utilization Playground".

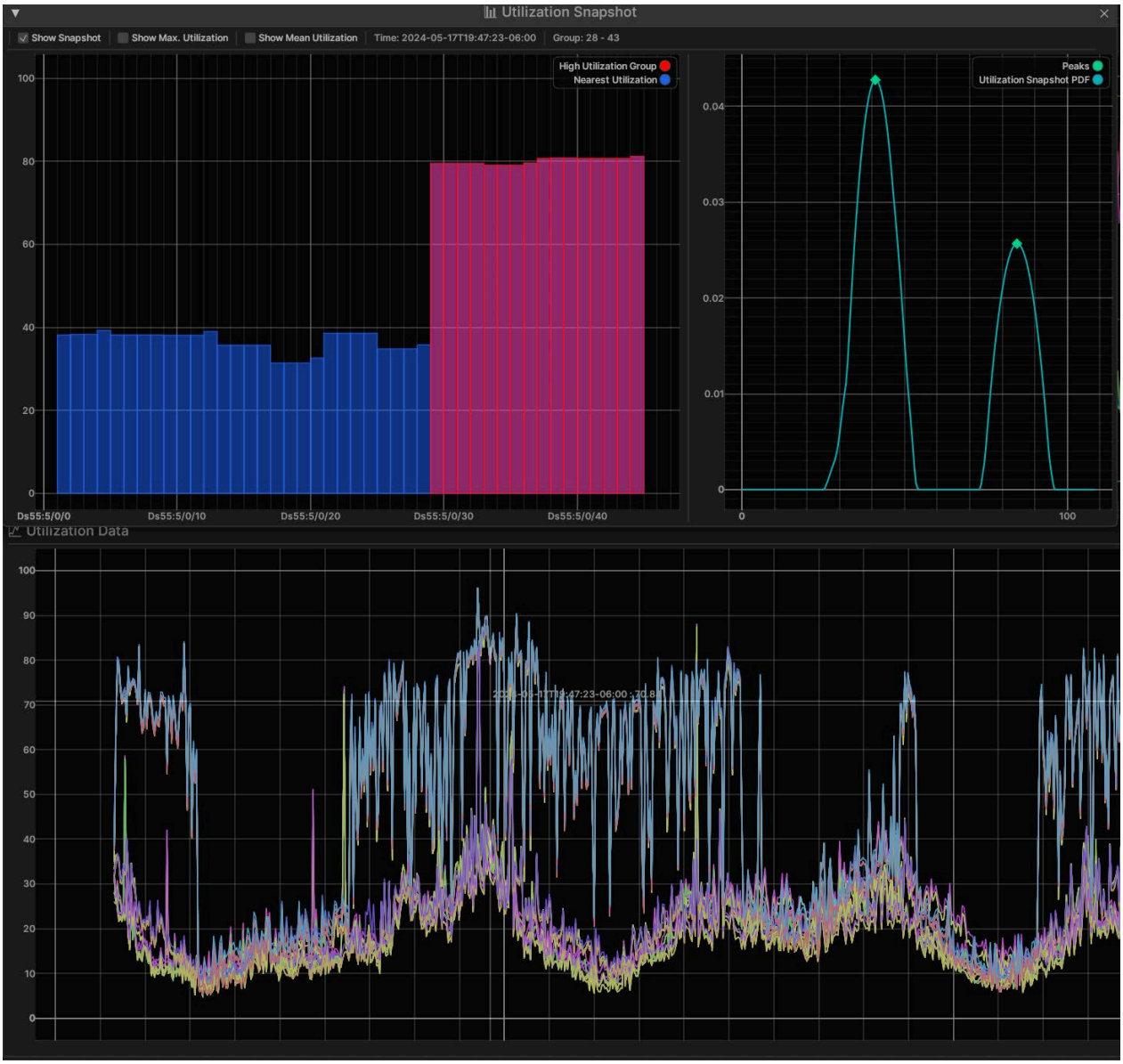

Figure 16 – Utilization Playground: Top Channel Set Detection

2.4. Key Performance Indicators

In addition to imbalance detection and top channel set detection, it is important to calculate KPIs for scoring and continuous monitoring.

The KPIs included in the library for imbalance detection are:

- The mean utilization of all samples of all channels.
- The 98th percentile of all channel samples' mean utilization values.
    - This provides insights into the utilization that a service group will experience within the busiest 30 minutes during the day.
- The mean utilization of all samples of the identified channel sets caused the imbalance.

  - Comparing the top channel sets mean utilization and other channels' mean utilization indicates the degree of imbalance from mean utilization point of view.
- The 98th percentile of the identified channel sets' mean utilization values.
  - Comparing the 98th percentile utilizations of the top channel set, and other channels indicates the degree of imbalance from the busiest 30 minutes point of view. Note that the 98th percentiles do not necessarily align in time.
- Detected imbalance duration over the total data duration ratio.
  - This ratio helps prioritize the detected channel sets. Commonly, the imbalance is caused by the "top channel set" identified using this ratio.
- Fairness calculation results:
  - Mean of all snapshot fairness values over time.
  - Mean of snapshot fairness values within detected time windows.
  - Coefficient of variation of mean channel utilizations over time.
  - Coefficient of variation of mean channel utilizations within detected time windows.
  - Mean standard deviation of the snapshot utilization values over time.
  - Mean standard deviation of the snapshot utilization values within detected time windows.
- Channel set detection results: a list of channel sets.
  - The channel sets are identified by their start channel index and end channel index.
  - The detected sample counts are calculated and included for each channel set.

2.5. Algorithm Performance

A performant software program allows for scaling at low cost with high confidence. For reusable and easily testable code such as imbalance detection, top channel set detection, and the KPI algorithms, performance optimization often comes with high return at a reasonable development cost. In addition, benchmark tests can help track performance changes in relation to the code changes and are a valuable addition to the algorithms' test suite. Below are the current benchmark results:

Table 1 – Imbalance Detection, Top Channel Set Detection, and KPI Calculation Benchmark Results (Single-Core, Confidence Level = 95%)

Test	Lower Bound	Mean	Upper Bound
Imbalance Detection, Top Channel Set Detection, and KPI Calculations (44 channels, 7-Day Utilization, 5-Minute Interval)	47.536 µs	48.159 µs	48.913 µs
Time Window Intersections, 1,000 x 1,000 Windows, $O(max\{m,n\})$	14.226 µs	14.619 µs	15.111 µs
Time Window Intersections, 10,000 x 10,000 Windows, $O(max\{m,n\})$	169.61 µs	202.76 µs	246.71 µs
Time Window Intersections, 100,000 x 100,000 Windows, $O(max\{m,n\})$	1.3363 ms	1.3697 ms	1.4078 ms

For our use cases, the imbalance detection and KPI algorithms are sufficiently performant and allow for large scale computing at a low cost - especially considering the nature of decomposed workloads in a vCMTS environment where this compute resource is completely independent of the data plane resource

responsible for forwarding and scheduling traffic. This can be more difficult in an I-CMTS compute environment.

Therefore, data I/O will be the most significant bottleneck (as in many other software programs) for imbalance detection and KPI calculations.

2.6. Per-CM Usage Data

An important prerequisite for utilization-based load balancing is per-CM usage over-time data. This data is used by the load balancing algorithm to weight users' impact on their devices' compatible channel sets and to assign bonding groups to the devices. As a cost-effective solution, we use the periodically collected and stored 64-bit interface octet counters from CMs to calculate their bandwidth usage on each channel. A basic visualization of the collected data is shown in Figure 17.

At least 3 days of CM usage data is used for calculating the aggregated "scores" for the CMs. The common aggregation methods are:

- Over-time bandwidth usage mean.
- A certain quantile (e.g.: 0.8) of the over-time bandwidth usage.
- Mean/quantile of the usage samples during peak hours.
- Mean/quantile of usage samples weighted by time-of-day factors.

In addition to the listed aggregation methods, machine learning based methods can predict user activity scores for the CMs. There are several options to choose from, but they do not guarantee improvements. The details of this approach are outside of the scope of this paper.

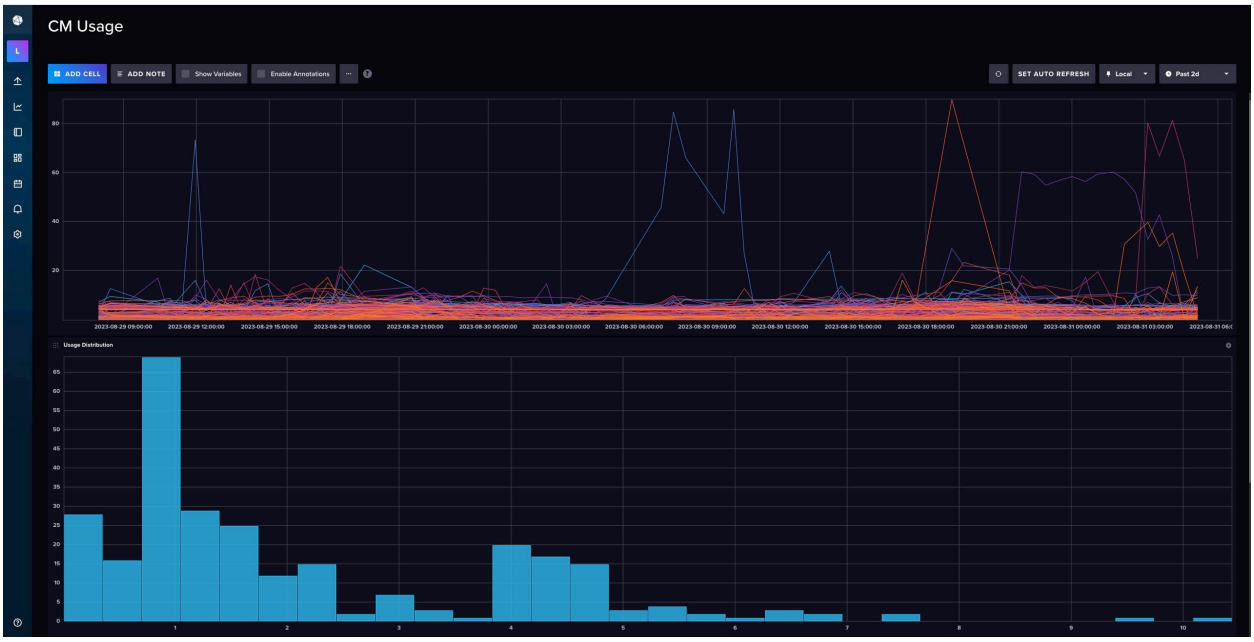

Figure 17 – Per-CM Usage Over Time and Mean Usage Distribution Calculated from Interface Octet Counters

3. Load Balancing

With imbalance detection and per-CM usage data, we designed and implemented our utilization-based load balancing algorithm. These algorithms were specifically developed to satisfy our use case requirements without referring to any existing DOCSIS channel load balancing algorithms.

3.1. Load Balancing Context and Requirements

Fundamentally, the external load balancer is made possible and viable by a foundational technology – the vCMTS. The vCMTS provides operational flexibility, observability, and APIs for distributing its control functionalities anywhere. Hence, the context and abilities of the external load balancer are vastly different than that of the traditional load balancers.

A traditional built-in dynamic load balancer does not have access to information of device vendor, model, firmware version, and long term per-CM usage data. Therefore, it relies on and reacts to short-term data and limited information to make dynamic improvements. Lack of full data perspective often results in challenges such as:

- Over-correction or under-correction by relying on short-term data, resulting in unideal load balancing outcome.
- "Discovery" of unknown service-affecting CM bugs that only exist on certain device models or firmware versions because the dynamic bonding changes (DBCs) are applied in a model/make agnostic fashion.
- Frequent DBC changes decrease the stability of the service.
- No observability for the load balancing decision making and reasoning.
    - No knowledge of the load balancing KPIs, and "before" and "after" KPI changes for feedback.
    - The load balancing decision is not explainable as the internal weighting and processing of the per-CM utilization data is unknown to the operator.

In addition, a traditional built-in load balancer's software updates/patches are often packaged along with the entire software build for the CMTS. This closely ties the software release cycles of the load balancer to a much larger system and makes it challenging and time consuming to develop, release and test improvements based on immediate feedback from deployment.

With the understanding of the limitations of a built-in load balancer. The top functional requirements of the external load balancer are:

- Consuming long-term per-CM usage data to maximize spectrum efficiency gain.
- Accounting for flexible policy configurations that can easily be tuned over time and device vendor/model/firmware information in the load balancing algorithm.
    - The bandwidth demands of devices excluded by policy must be accounted for by the algorithm, but their bonding groups must not be changed.
- Making recommendations and load balancing changes at low frequency to minimize the number of dynamic changes in the network.
    - At most once per day in a service group.
    - Balancing the load only when imbalance is detected.
- Tracking load balancing recommendations with associated input data for visibility and potential rewinding analysis for troubleshooting.
- Being distributed, decoupled from the CMTS software build, and elastically scalable.

These requirements provide reasoning for the design of the algorithm for the external load balancer.

3.2. Load Balancing Algorithms

This utilization-based load balancing algorithm consumes the user activity levels as scores. This allows the load balancer to take advantage of various aggregation methods implemented by the per-CM usage data service. This also allows the load balancer to be used as a CM count-based load balancer when per-CM usage data is not available, and the load balancer is supplied with a static score of 1 for all CMs.

3.2.1. Utilization-Based Load Balancing without Bonding Group Changes

The first load balancing algorithm designed and implemented uses the existing, configured bonding groups for the service group. This allows the bonding group recommendations produced by the external load balancer to be seamlessly applied without making service group level configuration changes that are more complex to perform without service impact.

This algorithm performs the following steps:

1. Put the existing bonding groups into distinct categories by bonding group size (for different CM capabilities), each category maintains a max binary heap where the order of the bonding groups is always maintained by their available resources (in descending order).
2. Sort the CMs based on their per-channel usage in descending order. Those have high per-channel usages are more likely to cause imbalance and are allocated first.
3. Iterate through all CMs along with their activity level data.
4. Match a CM to the bonding group set (candidate bonding groups) by its bonding capability.
5. Pop from the heap that is mapped from the bonding capability, this suggests a bonding group with the most available resource within its category.
6. Assign the bonding group to the CM and update the bonding group's available resource metric.
7. Put the bonding group back to the heap, the order of the bonding groups in the heap should automatically adjust.
8. Continue step 3 – 6 for all CMs.

We use this effective and performant algorithm as the default selection for the external load balancer. Its time complexity is $O(n \cdot log\, n)$, where n is the number of CMs, and its high-level performance measurements are listed below.

Table 2 – Load Balancing Algorithm 1: Benchmark Results (Single-Core, Confidence Level = 95%)

Test	Lower Bound	Mean	Upper Bound
600 CMs	446.31 µs	455.72 µs	466.88 µs
1,000 CMs	713.97 µs	720.86 µs	728.83 µs

Since the number of selected target service groups is small after imbalance detection, this algorithm is sufficiently performant. Especially considering the nature of workloads in a vCMTS environment where this compute resource is completely independent of the data plane resource and becomes a question of compute cost and associated value as opposed to competition for scare compute resource.

3.2.2. Utilization-Based Load Balancing with Bonding Group Recommendations

Alternatively, we designed another load balancing algorithm that recommends bonding groups for optimal load balancing results. This embraces the fact that changing the configurations of bonding groups on a service group may incur significantly higher complexity for hit-less, dynamic operations.

This algorithm performs the following steps:

1. Take the list of CMs along with their usage and bonding capabilities as the input.
2. Sort the CMs based on their per-channel usage in descending order. Those have high per-channel usages are more likely to cause imbalance and are allocated first.
3. Maintain an array of values that represent each channel's usage score, starting from 0.
4. Iterate through every CM and use a moving window that matches the CM's bonding capability to identify a contiguous set of channels that has the least usage and meets the primary channel requirement and policy constraints; water-fill the CM's usage to the selected set of channels.
5. Step 4 creates a set of bonding groups and bonding group assignments for improved load balancing. Note that the upper limit of the number of recommended bonding groups should be accounted for considering the gain, complexity, and system overhead. To enforce the maximum number of bonding groups limit:
    a. Sort the generated bonding groups by the number of CMs assigned to them.
    b. Iterate through the bonding groups, reduce the number of bonding groups to the limit number if needed, while ensuring all CM bonding capabilities are supported.
    c. Collect the remaining set of bonding groups after the previous reduction step.
    d. Re-execute step 3 with an additional constraint where the candidate bonding groups are the remaining bonding groups produced by the previous step.

One advantage of this algorithm is that it can potentially be used to reduce the number of bonding groups in the service group configuration, reducing the complexity of the configuration when necessary.

3.2.3. Device Exclusion Policy and Implementation

The device exclusion policy ensures the load balancing recommendations accommodate potential risks of DBC instability of certain CM models. Meanwhile, the load balancing algorithm must ensure that the utilizations of all devices including the excluded devices are considered. Both previously discussed load balancing algorithms can apply device exclusion policy. In the first algorithm, this is done by adding the following steps before the step 1:

1. Identify CMs to be excluded from load balancing by matching their model, make, bonding capability, and firmware version.
2. Maintain a hash map with key being the bonding group name and the value being the bonding group instance; for each excluded CM, instantiate and insert its bonding group to the hash map if it has not been inserted previously, and subtract its score from the bonding group’s available resource.
3. Combine the bonding groups initialized in the previous step with other existing bonding groups (with default initialization) in step 1 of the first load balancing algorithm.

Similar steps are inserted between the step 3 and 4 of the second load balancing algorithm to ensure the bonding groups used by the excluded CMs always exist, and the usage scores of excluded devices are considered after the initialization of the channel usage scores. These exclusion policies that can change

over time as modem firmware fixes are obtained and certified are also much easier to manage as an external micro-service in a virtualized CMTS.

4. Cloud-Based Load Balancing Service

4.1. Architecture

Utilizing all previously discussed algorithms, we developed a cloud-based load balancing service for detecting and monitoring utilization imbalance and improving spectrum efficiency with load balancing recommendations.

To ensure optimal performance and robustness and minimize the cost of this cloud-based load balancing service, we chose to use Rust to develop the algorithms and most of the distributed components. It allows us to be meticulously explicit about the system's behavior and boundaries and have high predictability in all aspects. The system components and their high-level relationships are illustrated in the block diagram in Figure 18.

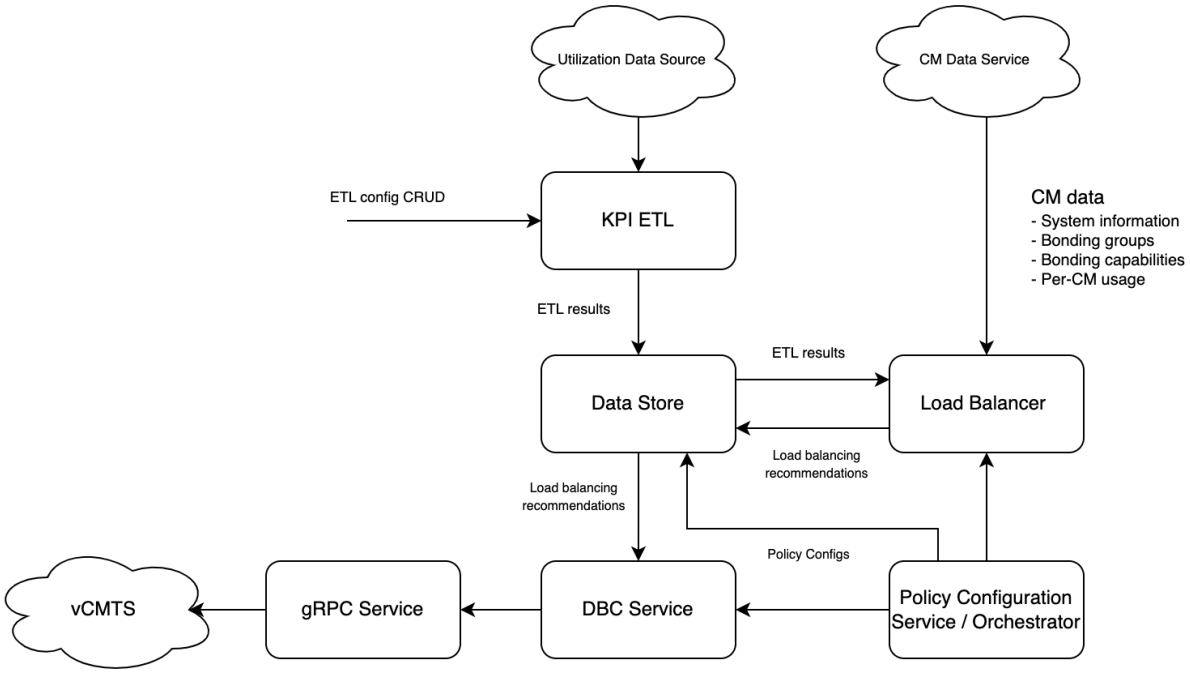

Figure 18 – External Load Balancer: System View

It is worth noting that the external load balancer's computing demand is extremely low, optimizing compute cloud cost, when it is scaled for full production. Therefore, for the system's simplicity, the design is in favor of not introducing task queues or message brokers for horizontal scaling in the near future. Even if more computing resources are needed, each of the components can be easily replicated and their data can be partitioned for scaling.

4.2. System Components

- KPI ETL
  - The KPI ETL service is responsible for detecting imbalanced loads across production service groups by consuming per-channel utilization time series from the utilization data

source. The KPI ETL uses the imbalance detection algorithm, top channel set detection algorithm, and KPI calculation methods discussed in Section 2.
    - The configurations for KPI ETL specify the scope, schedule, and enabled service groups. The results produced by this service contain flags indicating if the service groups' channel utilizations are imbalanced. This flag, along with other identifiers are used in the query to KPI ETL's API for selecting imbalanced service groups at different network topology levels.
- Load Balancer
    - The load balancer provides APIs for the load balancing algorithm and for triggering load balancing on certain service groups with policy configurations and create, read, update, and delete (CRUD) support for load balancing results in the data store. The load balancer also integrates with the APIs of the CM data service for reading CM data for the load balancing algorithm.
- Utilization Data Service
    - The utilization data service is a time series database (TSDB) that provides APIs for the KPI ETL to read over-time data including per-channel utilization at 15-second intervals. Its API parameters support the caller to specify the time duration, and step size of the data.
- CM Data Service
    - As discussed in Section 2.6, the CM data service provides per-CM usage data based on samples collected over time for each device. It also implements data aggregation methods for preprocessing and summarizing per-CM usage data for the load balancer.
- External Load Balancer Data Store
    - This is a data store that maintains KPI ETL results, load balancing recommendations, and policy configurations etc. It ensures atomicity, consistency, isolation, and durability (ACID) in transactions and low frequency information updates for services and between services. It is not limited to a single instance of database.
- Policy Configuration Service and Orchestrator
    - The policy configuration service and the orchestrator provide facade for external load balancer services, CRUD APIs for policy configurations, and scheduling APIs and functionality for the calculations.
- DBC Service
    - The DBC service transforms the load balancing recommendations into batched, parallelized remote procedure calls (RPCs) to the vCMTS service gateway for changing the CMs' bonding groups with throttling, error handling, and scheduled retries in place.
- Monitoring System
    - At the high-level, the monitoring system consists of TSDBs, Prometheus endpoints provided by the services, and Grafana dashboards. An example metric scraped by Prometheus from KPI ETL is shown in Figure 19.

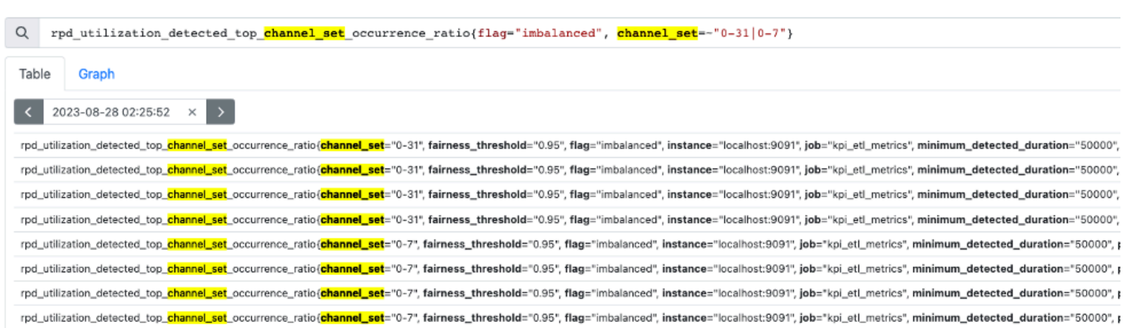

Figure 19 – Monitoring: An Example KPI Metric

5. Load Balancing Results

We are in the early stage of exercising closed-loop DBCs and validating the recommendations in our lab and field trials for this cloud-based external load balancer. And we have conducted experiments for load balancing with several detected service groups. All selected service groups consist of mixes of DOCSIS 3.1, DOCSIS 3.0, and DOCSIS 2.0 devices. In this section, we discuss the "before" and "after" utilization distributions and the improvements provided by the two load balancing algorithms. The improvements we have seen in imbalanced service groups by KPI, and visually in the figures where each color represents a modem, in this section are significantly improved by both algorithms described above.

In the "before" and "after" plots, each device's aggregated usage is water-filled to their bonded downstream SC-QAM channels, this emulates the effect the downstream scheduler can produce. In the experiments, the usage aggregation method is the 80th percentile in each CM's over-time usage samples, and only the samples within time windows where the mean utilization of all downstream SC-QAM channels is greater than or equal to 30% are selected. This aggregated value uses megabits per second (Mbps) as its unit as shown in the plot's x-axis in Figure 20. The plot's y-axis represents the downstream SC-QAM channel IDs starting from 0. And each color in the plot represents the usage from an individual device with the "water-filling" effect.

In the "before" snapshot of the service group 1 shown in Figure 20, several high-activity users are identified, and the overall utilization is not uniformly distributed. This is a fundamental limitation of the static load balancer where it does not take user demand variations into account. For service group 1, the channel set 32 – 35 was detected as the top channel set, and a primary contributor of the imbalance is a highly active DOCSIS 3.0 4x4 CM. After optimization, both of our load balancing algorithm 1 and 2 projects uniformly distributed usage across all 44 downstream SC-QAM channels as shown in Figure 21 and Figure 22. As for the KPI, the fairness value significantly increased after the optimization. Note that after moving the CMs, their colors do not necessarily stay the same. In the analysis for this experiment, the CM media access control (MAC) addresses are used for device identification.

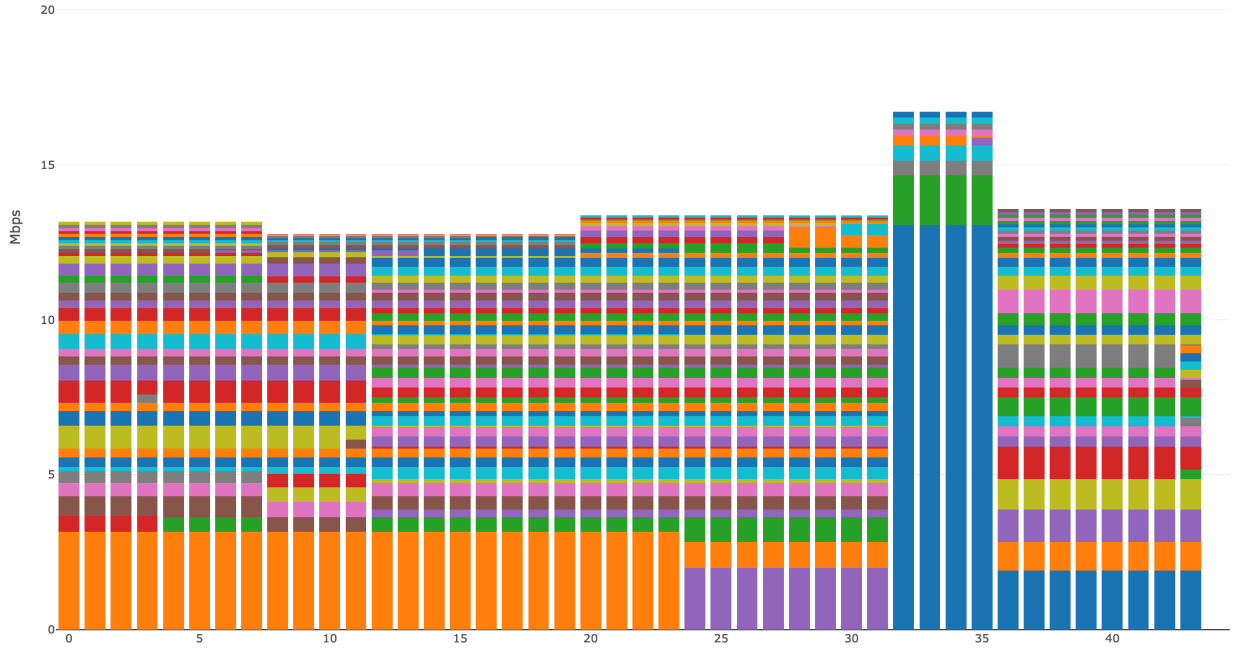

Figure 20 – Service Group 1: Before

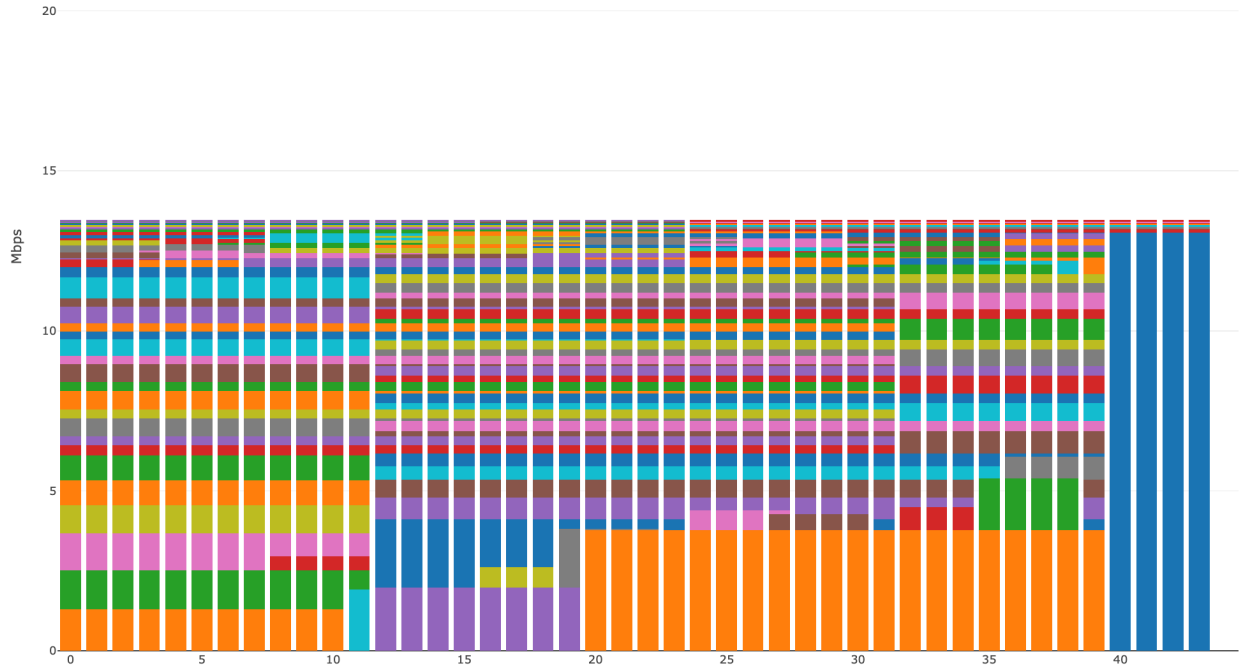

Figure 21 – Service Group 1: Optimized by the Algorithm 1

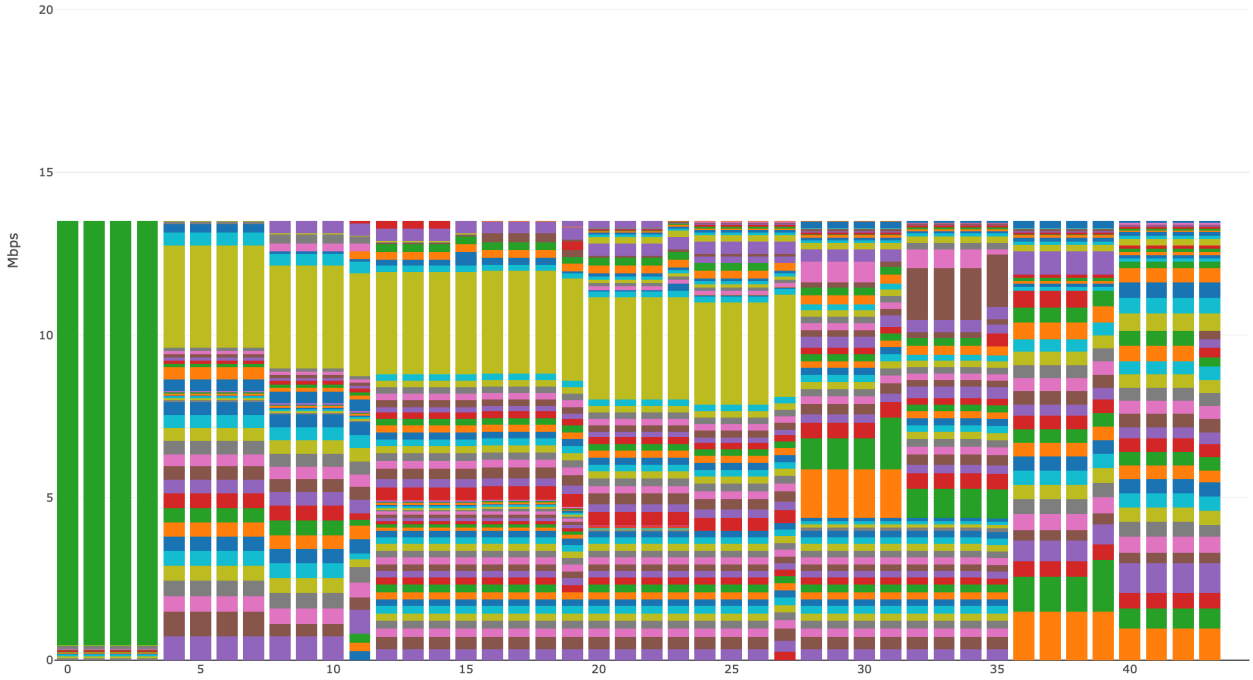

Figure 22 – Service Group 1: Optimized by the Algorithm 2

In the “before” snapshot of the service group 1 shown in Figure 23, the channel set 0 – 31 is being detected as the top contributor to the imbalance, and there are several highly utilized devices. The load balancing was significantly improved after applying either algorithm 1 or 2, as shown in Figure 24 and Figure 25.

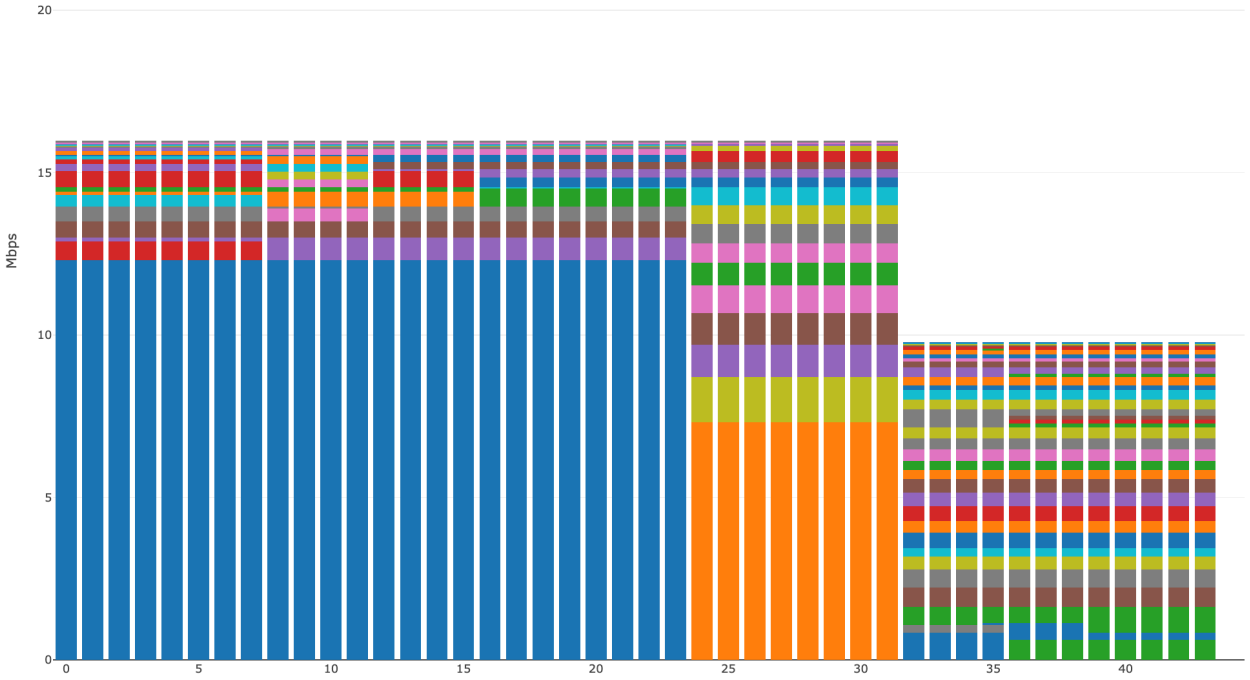

Figure 23 – Service Group 2: Before

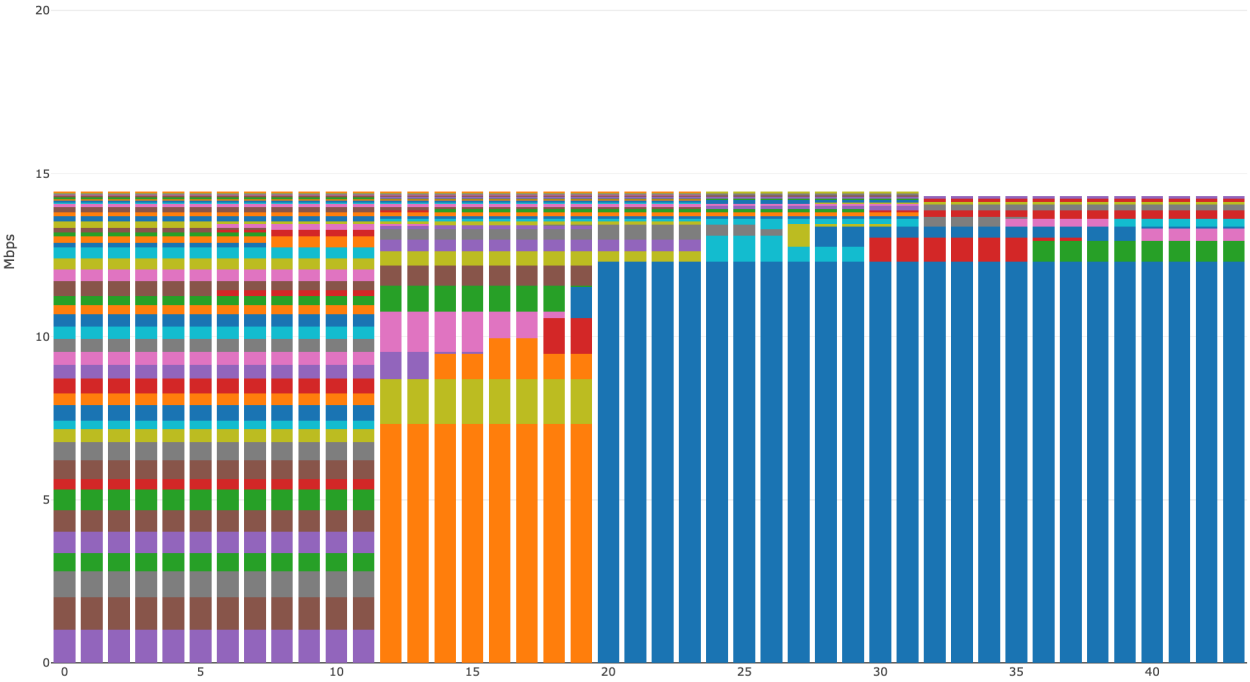

Figure 24 – Service Group 2: Optimized by the Algorithm 1

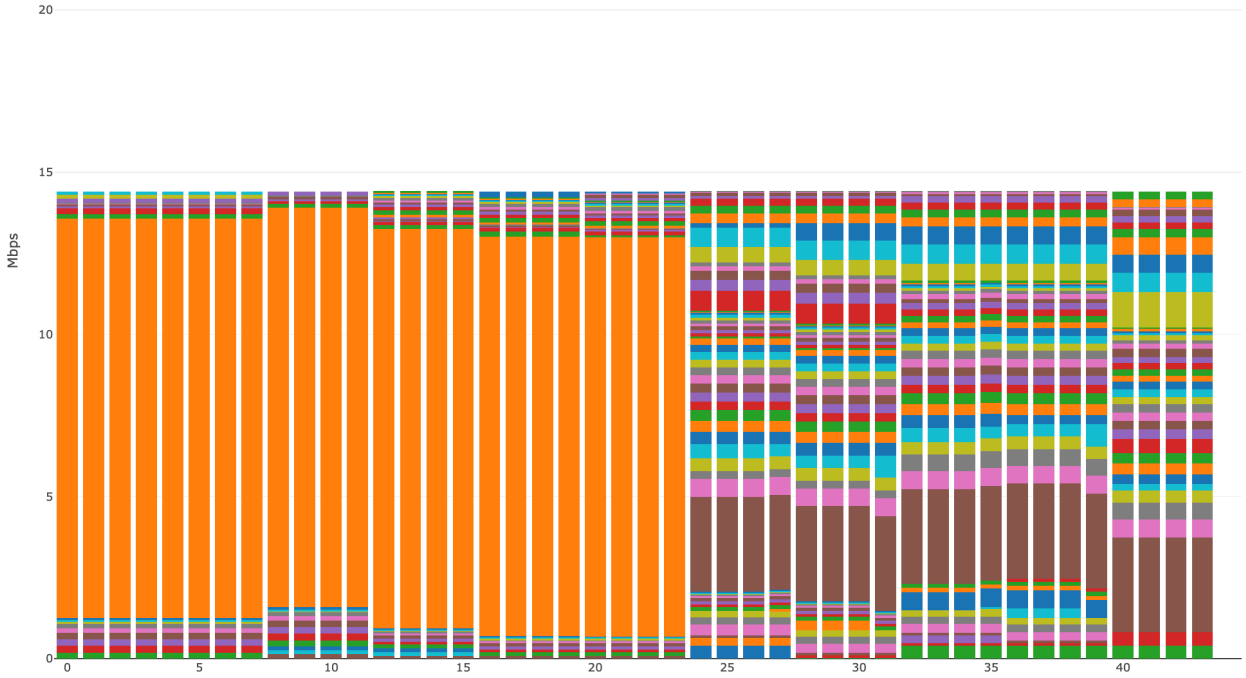

Figure 25 – Service Group 2: Optimized by the Algorithm 2

Finally, the service group 3 started with 2 highly utilized 4-channel bonding groups on both ends of the SC-QAM spectrum as shown in Figure 26. This situation was able to be improved by both of our load balancing algorithms as shown in Figure 27 and Figure 28.

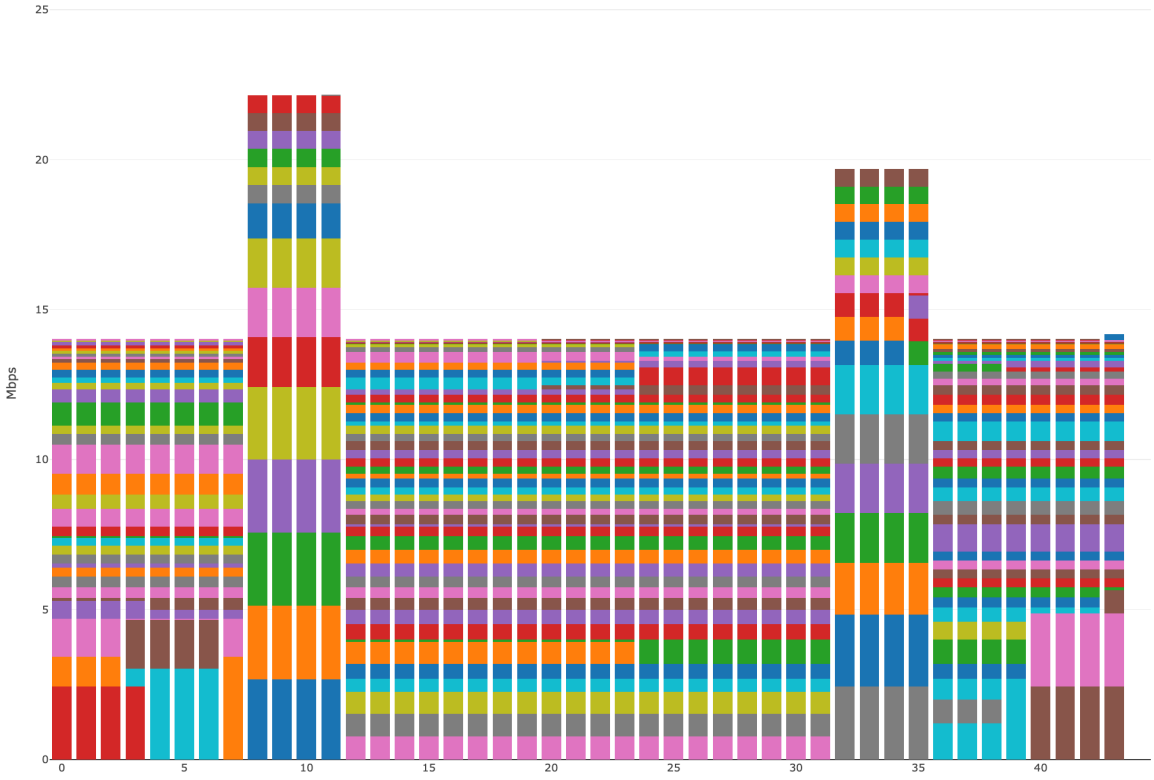

Figure 26 – Service Group 3: Before

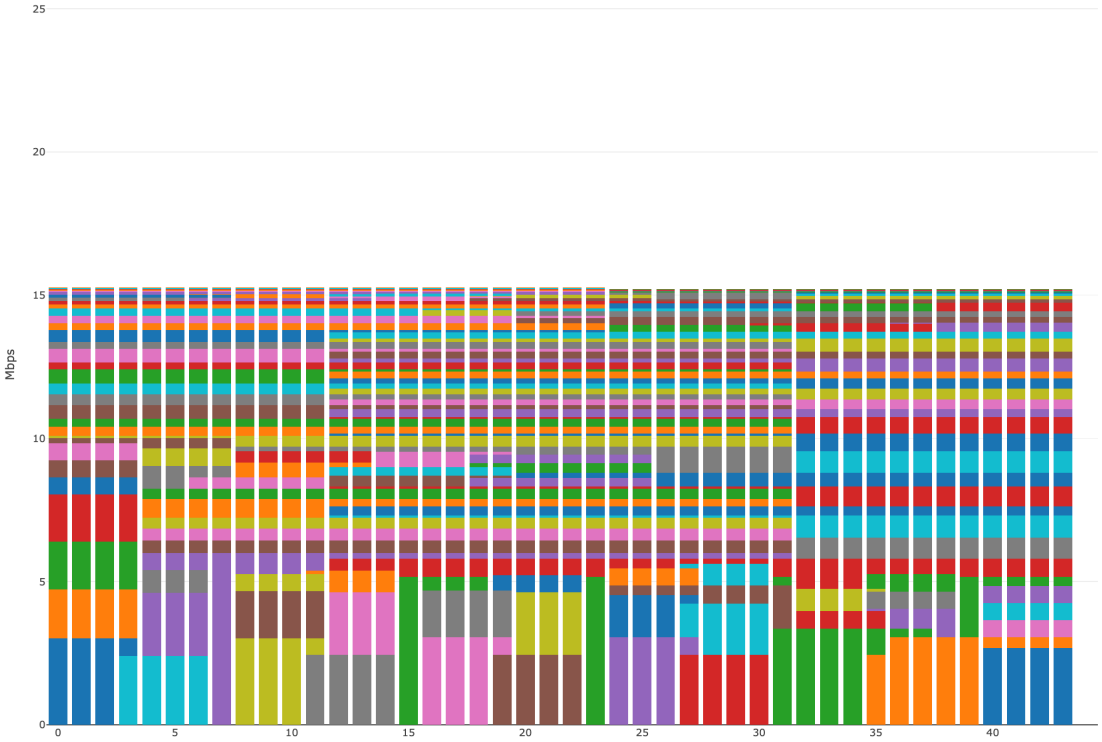

Figure 27 – Service Group 3: Optimized by the Algorithm 1

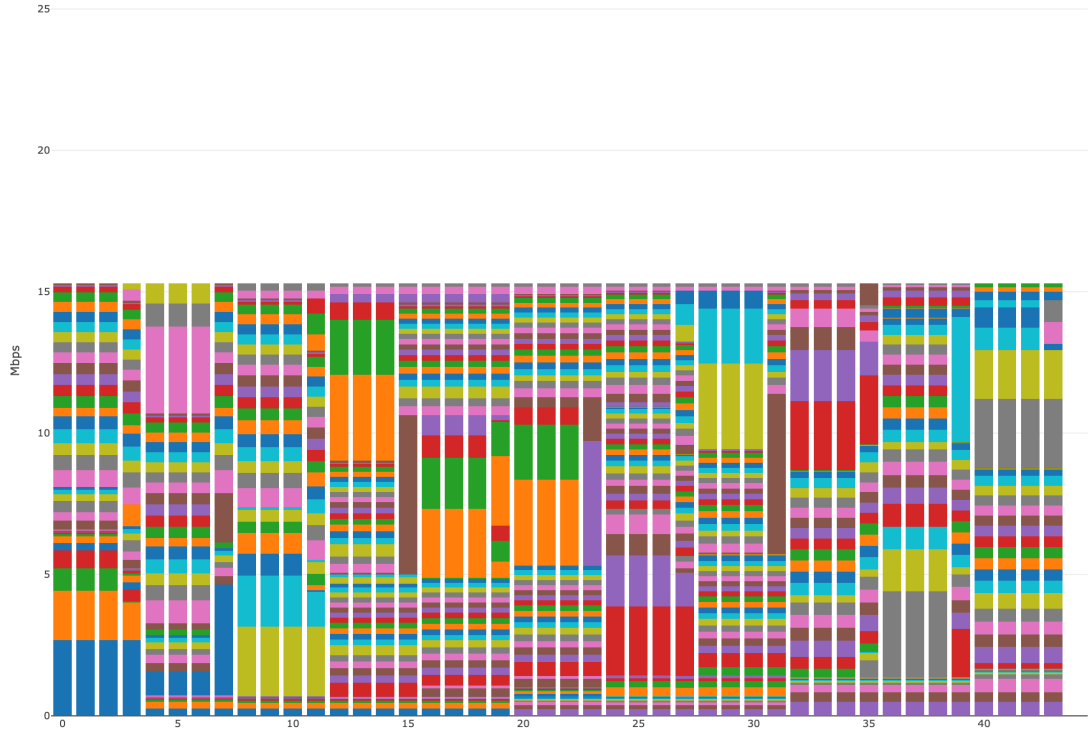

Figure 28 – Service Group 3: Optimized by the Algorithm 2

6. Conclusion

In this paper, we presented our motivation, observations, algorithm design, system design, and implementation of a cloud-based external load balancer for optimizing spectrum efficiency over the existing, built-in load balancers at a low cost. From the software system point of view, decoupling this functionality from the vCMTS not only simplifies the vCMTS components, but also shortens the load balancer's development, test, and release cycles for its functionality improvements. These algorithms and associated improvements would be difficult to implement in an integrated I-CMTS system without elastic cloud compute. In addition, the load balancer is empowered by rich datasets and information for it to make optimal bonding group recommendations, resulting in increased spectrum efficiency, increased capacity "headroom" for peak speeds, improved service reliability, reduced operational costs, and overall improved user experience.

Finally, this cloud-based external load balancer reinforces the model of building virtual network functions (VNFs) to take actions based on data analytics results to improve our service quality and resource efficiency at low cost and low risk in a fully automated fashion – examples of such VNFs are the spectrum manager which was presented at SCTE 2023 [4] and our profile management system presented at SCTE 2023 [5]. In the large picture, such VNFs transform our network into smart, reliable, self-optimized, and self-healing systems.

Abbreviations

ACID	atomicity, consistency, isolation, and durability
API	application programming interface
CM	cable modem
CMTS	cable modem termination system
CRUD	create, read, update, and delete
DBC	dynamic bonding change
DOCSIS	data over cable service interface specifications
ETL	extract, transform, and load
GUI	graphical user interface
I/O	input/output
I-CMTS	integrated cable modem termination system
KDE	kernel density estimation
KPI	key performance indicator
MAC	media access control
Mbps	megabits per second
OFDM	orthogonal frequency division multiplexing
OFDMA	orthogonal frequency division multiple access
PDF	probability density function
RPC	remote procedure calls
SC-QAM	single carrier-quadrature amplitude modulation
SCTE	Society of Cable Telecommunications Engineers
TSDB	time series database
vCMTS	virtual cable modem termination system
VNF	virtual network function

Bibliography & References

1. *DOCSIS 4.0 MAC and Upper Layer Protocols Interface Specification*, CM-SP-MULPIv4.0-I08-231211.

2. Jain, R., Chiu, D. M., & Hawe, W. (1984). *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems," Digital Equipment Corporation*. Technical Report DEC-TR-301.

3. *David W. Scott* “Multivariate Density Estimation: Theory, Practice, and Visualization” (1992), Wiley Series in Probability and Statistics. Publisher: Wiley. ISBN: 9780470316849.

4. *Towards Fully Automated HFC Spectrum Management,* J. Zhu, M. Harb, J. Howe, C. Humble, and D. Rice, NCTA technical paper, 2023.

5. *Deploying PMA-Enabled OFDMA in Mid-Split and High-Split*, M Harb, D Rice, K Dugan, J Ferreira, R Lund, NCTA Technical paper, 2022.

