SCTE·ISBE CABLE-TEC EXPO 2019 NEW ORLEANS, LA SEPT. 30-OCT. 3

2019 Fall Technical Forum SCTE·ISBE • NCTA • CABLELABS

Kickstarting Proactive Network Maintenance with the Proactive Operations Platform and Example Application

An easy way to start your own PNM journey

A Technical Paper prepared for SCTE•ISBE by

Jason Rupe, Ph.D.
Principal Architect
CableLabs®
858 Coal Creek Circle
303.661.3332
j.rupe@cablelabs.com

Jingjie Zhu
Senior Engineer
CableLabs®
858 Coal Creek Circle
303.661.3312
j.zhu@cablelabs.com

Table of Contents

List of Figures

Introduction

As part of its long standing proactive network maintenance (PNM) project, CableLabs® has been assessing the needs of operators and vendors in the area of PNM with the goal of reducing adoption friction for members and vendors. We identified a few main issues, an important one of which is addressed by the work presented in this paper: the Proactive Operations (ProOps) platform.

ProOps is a platform (environment, framework) for turning data into operations action. That includes proaction, when the data allow it. PNM data enables proaction, so we built an example application that comes with ProOps, which serves multiple purposes: as an example to show how to use ProOps, as a starting point for trying basic network data-driven PNM and reactive operations, and as a launch point for implementing and sharing PNM best practices.

To turn data into action, most operators rely on engineering and technician expertise. It is common to simply gather and plot the data, then look at the output. CableLabs built the cable modem validation application (CMVA) for that latter purpose (as well as for cable modem (CM) certification test automation and sharing). But without a human expert sifting through the data, not much can be done with it. CMVA is great for developing PNM ideas, but it still requires experts to do the next step, and developers to build solutions to try. That requires investment risk that we surmise is a roadblock to implementation of PNM. But for many operators, and some vendors, there just aren't enough available experts to do the work manually. The industry needs help getting over the hurdle of turning the data we exposed into action we can take with confidence.

ProOps was built to facilitate the automation of turning data into operations action. Generally, we identified the steps to accomplish that task as 1) data extraction (observation), 2) analysis across time and network elements (orient), 3) correlating problems and measuring severity (decide), and 4) defining work items that are worthy of attention (act). The steps can be labled as observe, orient, decide, and act (OODA) to roughly follow the OODA process or OODA loop, which is a cyclic process developed by US Air Force Colonel John Boyd [1,2]. Combat operations resembles network operations more than we care to admit perhaps, so the labels fit. Boyd systematized the combat operations process as a rapid cycle of the OODA loop. Likewise, network operations follows a similar process, and the concept helps explain how ProOps works.

In the future, we expect to release applications and modules to enhance existing applications, in the ProOps environment. Once ProOps is installed, new applications will work like updates to ProOps, making the operations impact even less. New applications and modules will interwork with existing applications in the same deployment of ProOps, or parallel deployments utilizing the same common collection framework (CCF) instance are possible too. Multiple deployment models are available today, and more to come as vendor and operator members request.

ProOps is currently available for use by CableLabs members and vendors under nondisclosure agreement (NDA) and intellectual property rights (IPR), with the additional common code collection (C3) community agreement.

In the rest of this paper, we explain in-depth the structure and function of ProOps, and the example application that comes with it today. This will lead us to explain some of the basic ways you can configure and build your own solutions in ProOps to support your operations improvement experiments and inventions. We also cover use cases for ProOps that we envision, which hopefully will interest you or spur your own imagination to invent use cases we haven't thought of yet.

The Proactive Operations Platform

The ProOps platform is a framework for constructing applications that turn network data into action. ProOps is simpler to understand in the context of an application, so we later explain it in the context of the simple but complete example application that comes with ProOps. As a foundation, we first explain the ProOps application environment as it stands alone, before an application has been constructed. We also explain briefly here the elements of the environment, covering the details later in the application context.

An important class of components of ProOps is the worker, which is a module of code which can be tasked by the system. Some workers are a part of and come with ProOps, and some workers function for the application. A worker is a module of code which is scheduled to execute according to the configuration instructions, taking inputs as directed, and sending its output where directed. In software development terms, it is like a microservice, but subscribed to a task queue instead of waiting for an application programming interface (API) call. An application in ProOps is configured from workers that execute the needed instructions.

Figure 1 shows the elements of the PNM application environment as a raw environment, without hosting a working application, or any workers; the workers box represents just the worker organization without the workers that make up the application.

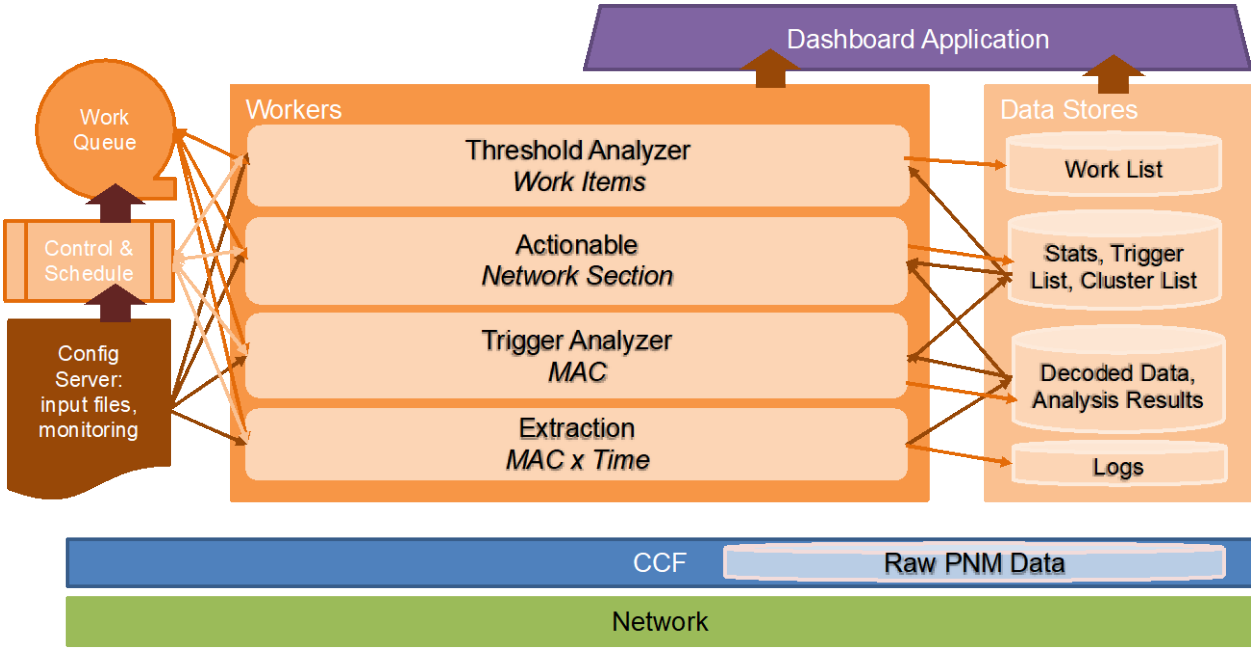

Figure 1 – A depiction of the ProOps platform on top of CCF and a network.

We begin at the bottom of ProOps and work our way up. The descriptions of the network or CCF are outside the scope of this paper. For the latter, we refer the reader to the CCF architecture document [3]. The remaining elements are all a part of ProOps, so are described briefly here.

- The configuration server (config server) hosts the configuration data which define the messaging mechanisms and necessary additional configurations for workers. It describes the workers to be connected, and any changes that affect the work they do. The user can control applications through this interface for the most part, as the configuration is what defines the application from a

pool of workers. When workers start up, they request configurations from the configuration server using RESTful APIs.

- The control and schedule section handles the scheduling of workers, and controls task creation and ending as needed to handle the processing according to the configuration server. The tasks describe the data that each worker is fed, the data that each worker outputs, where the outputs are held, where the inputs are obtained from, and any additional information for specific workers such as thresholds, Internet protocol (IP) addresses of cable modem termination systems (CMTSs) and CCFs, etc.
- The work queues are a number of first-in, first-out (FIFO) queues of tasks that need to be done by the workers according to the schedule dictated by the configuration server.
- The workers are held in the worker environment in the center of the figure. These workers simply exist in a pool and are organized as dictated by the configuration server, but we recognize that for PNM we will need the data from the network to be translated into actionable work. Therefore, we have identified reasonable steps we expect are sufficient for most any application envisioned. These steps are part of the design advantage of ProOps, but not a constraint to the solution possibilities. Those steps are briefly described as follows. Note how they align to the well known OODA loop or OODA process.
    - **Extraction** takes place periodically as dictated by the initial schedule outlined in the configuration. Because the same media access control (MAC) addresses are collected multiple times on a(n) (approximate) schedule, we note that this step conducts data collection of MAC by time **(MAC x Time)**. The output at this layer goes to the log file, but also decoded data and analysis results are held, too. Because this layer is focused on data extraction, we align this layer to the **Observe** step of the OODA process.
    - **Trigger Analyzer** will analyze the data from the extraction and determine whether a change of action is triggered, such as obtaining new data elements, taking data elements on a different schedule, collecting data from related network elements, and placing some of these MAC addresses on a list that identifies further action. The analysis on this layer likely evaluates data on a single MAC over time. For this reason, we note that this step conducts data analysis by **MAC**. The output from this step consists of statistics, and a determination of which MACs are triggered. Because this layer establishes some context to the information, we align this layer to the **Orient** step of the OODA process.
    - The **Actionable** step takes the MAC addresses that are flagged by the trigger analyzer and determines the network section relevant to the identified MACs. For example, clustering of MACs is possible here. But to make these actionable, we also have to introduce a measure of performance and comparison for these MACs or clusters of MACs, so that decisions can be made to bring the network section to actionable work. Therefore, we note that this step conducts analysis on the **network sections**. The output from this step can be clusters of MAC addresses, or single MAC addresses representing network points or CMs, but will definitely be additional statistics to support decisions. Because this layer makes our oriented view of the data something we can act on, it is really the layer in which decisions are made. Therefore, we align this layer to the **Decide** step of the OODA process.
    - The **Threshold Analyzer** will look at the measure of performance and comparison, introduce further analysis of other MACs in the cluster, look at historical information, and decide which of the identified actionable network sections are good choices to act on based on information provided in the configuration server. This step therefore conducts analysis on **work items**, and provides output to the work list. Because this layer considers the decision made that there is something that can be acted upon, and either selects or allows the selection of work that will be acted on, we align this layer to the **Act** step of the OODA process.

- The data stores hold the data elements needed for the application as dictated by the configuration server. The following elements are expected to be needed in many applications.
    - Logs will hold output from the extraction layer.
    - Decoded data and analysis results will be held in a data lake or database depending on the needs of the application.
    - Statistics, triggered lists, and clustering lists are captured in a database or tables as needed by the application.
    - The work list is likewise captured in a list or database as needed.
- Finally, a dashboard application is fed by the entire set of identified data, so that geographic information system (GIS) maps, graphs, and detailed work packages can be assembled as needed by whatever system this will feed. This approach allows most any operator to take the output from the various steps of an application in this environment and tie it easily into its existing ticketing systems and other operations needs.

The previous Figure 1 is revised now to show the flow of information and how the layers reflect the OODA steps, in Figure 2.

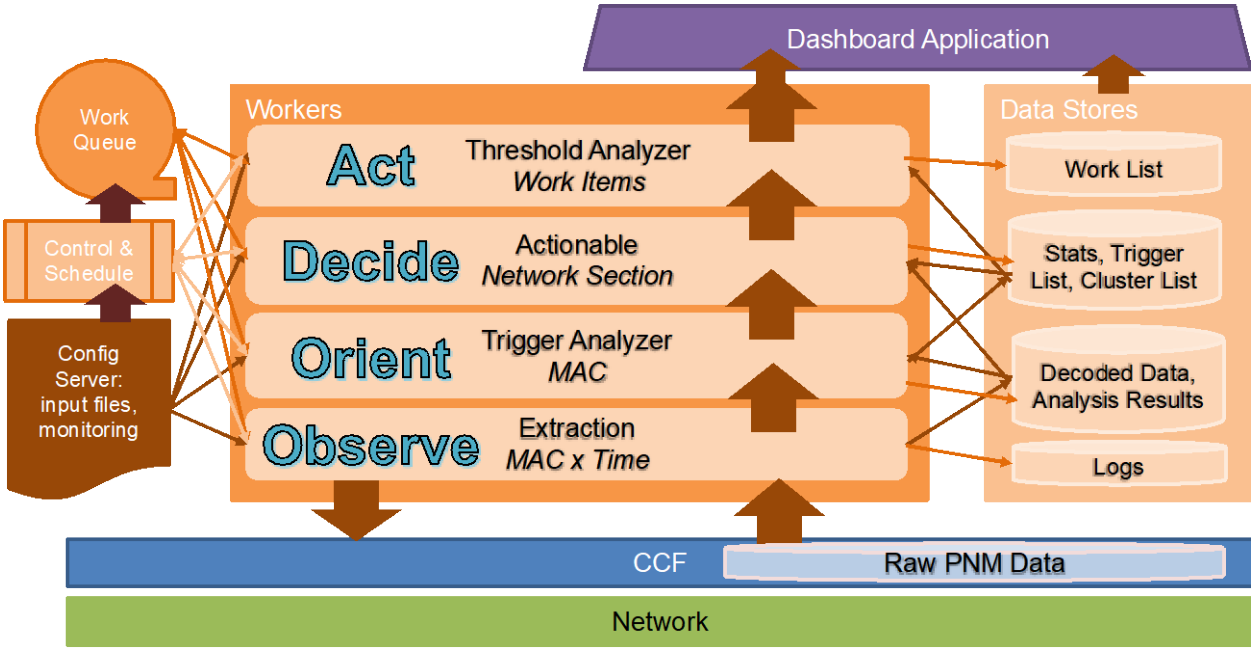

Figure 2 – The depiction of ProOps from Figure 1, with OODA overlaid and arrows showing the process flow.

Next, we discuss some of the detail in the sections of the ProOps architecture. In the explanation of the architecture, we put **workers in bold,** *data in italics,* and configuration underlined to help keep it all straight.

1. Control & Schedule Worker

Control & schedule is a special worker that interacts at the system level to control the other workers, follow the application and flow defined in the configuration file, and schedule work according to the task description or conditions discovered from the other workers or their data output. The tasks are posted periodically by the schedule worker to the task queue(s) and assigned to workers by subscription. If a

configuration instructs data collection to happen every *x* hours, then this worker will schedule work for the queue based on this schedule. Further, if a result from a data poll and analysis indicates triggering a rule given in the config file, then this worker will update the schedule it follows for those impacted CMs accordingly, and therefore change the schedule that dictates how it places work in the work queue. It must therefore hold state, but can always regain a lost state by reading the config data and the information in the data stores, as those are sufficient to define the schedule.

Note that if this worker is responsible for monitoring a trigger condition to instantiate another worker, such as a **ranker worker**, then it must have access to the data or worker output that triggers the follow up work. If instead it is to act on a schedule only, then maintaining the time against that schedule is important, but not catastrophic if state is lost. The *log file* could be used to capture time and a backup of **control & schedule** state to avoid a loss of state at failure.

This special worker also regulates the system so to not tax the network too much, and to poll and archive data based on scheduling requirements from the application configuration. **The control & schedule worker** reads the config data, builds a schedule for how other workers are started for data collection, then creates the needed polling workers at the appropriate time.

Note that the **control & schedule worker** is the one worker type that keeps track of timing; hence, it needs to maintain state information.

2. Work Queues

This entity is a number of queues of tasks to be handled by different workers in the system. The **control & schedule** worker will place tasks on the queues for other workers, and likewise another worker can generate a task for the **control & schedule** worker to handle. The work queues manage tasks and assignments, holding the tasks to do like a pipeline connecting tasks to workers.

3. Workers and Workflow Description

In this architecture, workers are created to handle tasks, and the work to be handled arrives to a queue. A special **control & schedule** worker handles all timing for scheduling, and assigns work to the other workers. Based on this concept, the following workers need to be defined, as illustrated in Figure 2. The previous figures only show in some cases the worker classes, so more detail is described here than can be shown in the figures. Further, other types of workers can be defined at will, and nothing precludes the addition of layers, or the extension of function of identified layers, in this architecture. The layering is simply a framework that facilitates action to support a network and services. The configuration file, described later, is what enforces the construct of the overall system, and therefore the workflows or applications built in the environment.

- Observe = Extraction [**polling worker, translator, linear calculator**]: This is a pool of workers which form a linear processing of data from the CCF. The new **polling workers** interact with the CCF to poll the data. The data received are then translated by **translator workers** if and when necessary, and finally processed by **linear calculator workers**. The linear calculator worker is a special calculator worker that is always applied to data right off of the CCF. The outputs are cataloged into the *decoded data and analysis results data lake or database* by a **catalog worker** (if needed).
- Orient = Trigger Analyzer [**calculator workers, anomaly detector workers, machine learning workers, trigger analyzer worker**]: This is a pool of workers who evaluate data as it shows up in the *decoded data and analysis results*, against the analyses invoked and configured by the config file, placing statistics and a list of triggered MAC addresses into the *Stats, Trigger List,*

Cluster list database. The **control & schedule worker** will monitor the database for new data which triggers the need for a specific worker type, or monitor a clock to trigger, based on the config file. Then it will schedule the appropriate workers to complete the configured tasks. **Calculator workers** will simply calculate statistics from the raw data and place the statistics into the database. **Anomaly detector workers** will search a series of data (over frequency, time, etc.), indicate the data included in a detected anomaly, and in some cases indicate the class of anomaly detected. **Machine learning workers** will apply other machine learning techniques to data and statistics. Both the latter worker types may contain or use calculator workers for example, too. A **trigger analyzer** worker will assess these outputs against the triggers indicated in the config file. The output from these analyses are tasks that can be to sample new data, or sample some data sources for specific MACs more frequently, or to re-analyze older data, for example.

> A note is warranted here about the role of the config file versus the changes decided: The config file holds information about the frequency of data collection, but only for the default data frequency; it also holds information about the triggers and actions from triggers regarding data polling frequency, but only as conditions, not as system state. Changes in system state are held in memory; or, in the case of MAC addresses subject to frequent polling, in the *stats, trigger list, cluster list* database.

This means workers can relate to all other workers in a nesting, series, parallel, or other configuration relationship. More worker types can be defined to fit into this pool as needed by specific applications.

- Decide = Actionable Layer [**severity calculator worker, ranker worker, cluster worker**]: This worker class likely only contains a few worker types (depending on the nature of the application configured), tasked with translating the trigger list and collected statistics and data for triggered MAC addresses, and providing the necessary performance measures with which to rank the MAC addresses. It can work exclusively with the *stats, trigger list database* as it reads the information, processes, then adds an updated measure of performance to the data, and provides an updated ranking. That assumes the configuration file is set up to provide the needed data to supply the measures needed for action. The work here can be done periodically, or triggered by a condition such as a number of data updates on the triggered list, or a number of new entries to the triggered list. The **control & schedule worker** will trigger action as directed by the config file, and trigger a **severity calculator worker** based on the information in the config file which specifies what triggers the calculation. Once the calculation is updated, the **ranker worker** will evaluate the updated measures of performance and rank the MAC addresses by severity, placing the performance measures and the ranked list back into the database. The **cluster worker** will evaluate the information relating to the MAC addresses and attempt to cluster them by similarities in the statistics and performance measures. Workers on this layer may need access to the raw data or decoded data or analysis results in some cases too. Either on a schedule or as triggered by conditions, as specified in the config file, the **cluster worker** is started to evaluate triggered MAC addresses and their performance and other needed information, conduct clustering based on multiple dimensions of data, and output clusters of MAC addresses that are likely experiencing the same problem.
- Act = Threshold Analyzer: The threshold analyzer will examine the trigger clusters (which are assumed to include clusters defined by single MAC addresses in many cases) and decide which of the clusters are actionable. This can be done through the config file which holds the rules that trigger the action based on an external financial model, operations rules, work load, number of potential jobs on the list, etc. The **control & schedule worker** will read the configuration file to determine when or under what condition to trigger a sorting of the opportunities in the *trigger cluster database* (and potentially the *trigger list*). The sorting is done by the **threshold analyzer**

sorter worker which will read the config file to determine what measures to use in the sorting, and then gather the information it needs, and finally sort the opportunities into a *dispatch list*. An output of this group of workers is also a measure of performance that is used for ranking, but potentially also additional measures for ranking or consideration including but not limited to an estimate of the benefit achieved by completing the associated work package. This analyzer layer can also call on external information such as available technicians in an area, day of week, likelihood of access to the network elements necessary, etc.

4. Config Files

One or more configuration files (config files) are necessary to define the application from the ProOps elements, and to control how the application functions day to day. There are several types of configuration information needed for this workflow, and that information could be stored in a single file, or multiple files, or in a database, for example. It could be edited in a file editor, or through a graphical user interface (GUI) defined for the application. The files could be separate or together or organized in any way. As developed currently, there is an interface with organization to configure workers and general parts of an application. We assume in this description that configuration will be read-in from a file, but will consider each configuration type as a separate file for the sake of explanation.

- Extraction Config File: This config file specifies the frequency of polling for each data element, and which data elements are polled as part of the application. It may also contain instructions for how to handle missing data, timeouts, and other problems that may be encountered in data polling.
- Trigger Analyzer Config File: This config file contains instructions of which analysis workers are included in the application or workflow, and how they are started (such as when data appears in the *decoded data and analysis results* database or data lake). It also includes the rules by which CMs are triggered for action to the next level, and flagged for ranking, plus what workers need to do to support the action layer.
- Action Config File: This config file contains the details of the model for the measure of performance for CMs based on their statistics and trigger state. The model may be complex or simple, depending on how data over time or missing data are treated, and whether a nonlinear model is needed. It also specifies what to use for ranking. This config file also may provide information for how to cluster CMs, essentially controlling the model for clustering, but also specifying the worker that is used.
- Threshold Analyzer Config File: This config file provides the rules by which opportunities (clusters or individual CMs) are ranked and sorted, included or excluded, on the dispatch list. It also specifies how often the list is revisited and updated, which means how often the clusters and individual CMs are analyzed.

5. Data Stores

Several different data stores are needed, though they need not be distinct. However, as the nature of what they store is different, it is likely advantageous to keep them in solutions suited best to their forms, and therefore desirable to be distinct in at least some way.

- *Raw PNM Data*: The PNM data can be kept in the file system of the CCF, or in the trivial file transfer protocol (TFTP) server used by the operator, or in a separate data lake. As the data held here are disparate, it is not likely that a database is an appropriate solution. Therefore we refer to the (Raw) PNM Data store as a data lake.

- *Decoded data, analysis results*: The decoded data and analysis results can go into a data lake or database, as long as the relations are held between MAC, time, and data to analysis. The data here can be contained in a relational database as it can contain decoded data and statistics of multiple types associated with individual CMs by type, time of day, etc.
- *Stats, trigger list, cluster list*: The data here can be contained in a relational database as it can contain statistics of multiple types associated with individual CMs by type, time of day, etc., and a separate related list of the CMs that meet the threshold for triggering based on the rules set in the config file. The cluster information can be held in a list, a database, or other simple form as it only needs to be a list of how the CMs are clustered in common groups for PNM work opportunities. But all these separate elements are related, so a relational database is a good candidate.
- *Work (dispatch) list*: This data store is simply a list of CMs or CM clusters which are opportunities for work. Each item on the list has a performance measure that indicates how critical the problem is, or how large the proactive opportunity is, so may even be an expected benefit in dollars, for example. There should be links back to the supporting evidence for the decision to add this work to a dispatch list, and to help with troubleshooting.
- *Logs*: The system will keep a list of system logs for overall system health. This may be a method to maintain state under failure too.

6. Scheduling Considerations

- A CM can usually only respond to one request at a time, or very few. But if a CM doesn't respond at the time data were requested, and there is reason to expect it should, then a retry may be in order.
- CCF jobs have three states: accepted, complete, and failed. Some complete jobs may have incorrect, errored, or incomplete data responses as well, so a complete job may still be a failed job.
- The parsing of the data output may result in a new fourth state of complete but errored.
- Thus, the state of a completed job from the CCF will create subsequent requests to be scheduled based on the final job status.
    - If the job status is complete, and parsing yields a correct output, then schedule the next job according to the instructions in the config file.
    - If the job status is complete but errored, or failed, then schedule a new request immediately, or after a short amount of time as specified by the config file.
    - The config file should also state whether subsequent jobs are scheduled based on the completion time of the last job, or based on a system clock. In other words, if a data type is requested six times a day, and the first one is delayed by an hour, then does the next data request happen in three hours or four hours? Both may be possible, but configurable in the config file.
- The **control & schedule worker** will need to sort jobs on a common system clock. The control worker handles this task.

7. Additional Notes about ProOps

- The config file must account for missing data in the way we analyze and calculate performance measures. A non-responsive CM may end up on a separate list.
- RabbitMQ is our choice for the first version of ProOps for managing the worker queue.
- We stagger polling to get the data we need and manage the poller resources.

PNM Example Application for ProOps

Here we define the example application that comes with ProOps. Out of the box, very little configuration is necessary to get the system running with the example application. However, the utility of the example application is highly dependent on the user's ability to configure it for their use case specifically.

The first version of the example application that comes with ProOps does not do clustering; the identified CMs will go directly to the threshold analyzer as single CMs. This can be reflected as each CM is its own cluster, and the cluster algorithm doing no work, but still following the diagram in Figure 3.

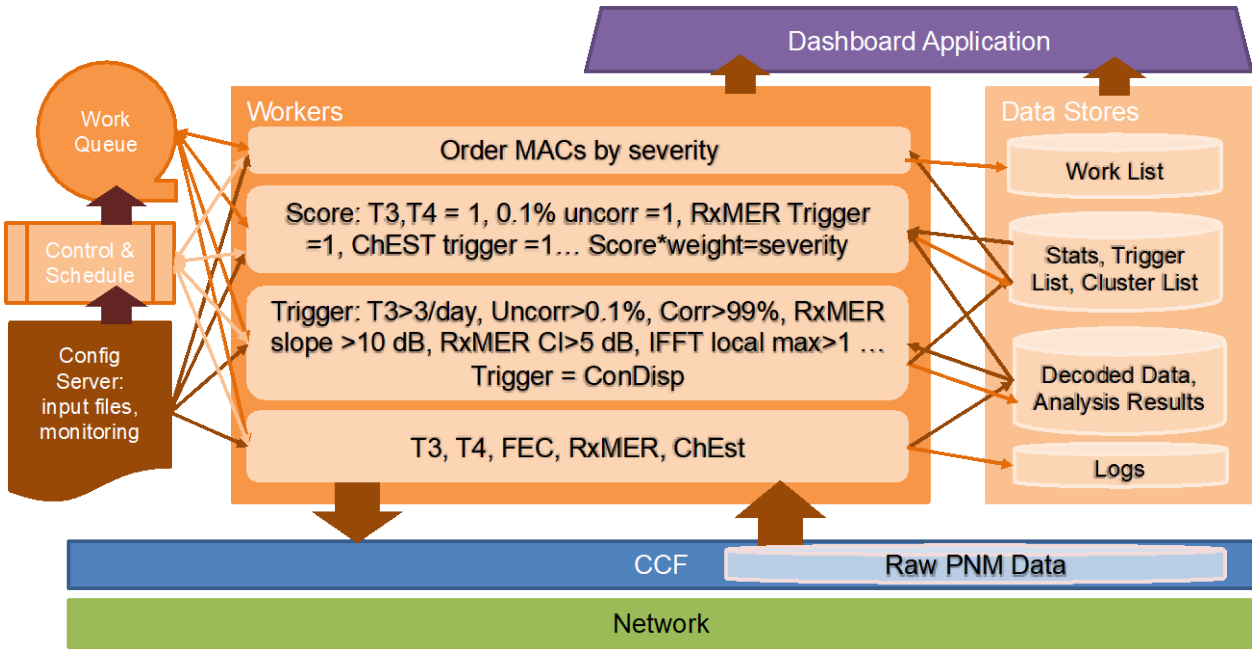

Figure 3 – Depiction of the example application that comes with ProOps today.

1. Base data polling (Pollers)

- Pull T3 and T4 errors from CM logs every hour (configurable).
- Pull forward error correction (FEC) stats every hour (configurable).
- Pull receive modulation error ratio (RxMER) every four hours (configurable).
- Pull channel estimation every four hours (configurable) at same time.

2. Triggers (Analyzers)

- Repeating errors at a configurable frequency, say more than three per day, configurable.
- Uncorrectables higher than 0.1%, correctables at 99% or higher for over an hour, all configurable.
- RxMER and SpecAn results with a slope greater than 10 dB, configurable. Also, trigger any RxMER with a 95% confidence interval greater than 5 dB (calculated from percentile responses),

configurable. Also look for ripples in either or both by inverse fast Fourier transform (IFFT) with an energy spike. Take the largest continuous segment of data from either and transform (IFFT) it into time domain, then search for local maximum. Each result with more than one local maximum is triggered (this removes the high frequency main energy). Transform to distance and severity of echo tunnel, trigger anything higher than 10 dB, configurable.

- ChEst indicating a range greater than 10 dB, configurable. Also look for ripples in either or both using same method as above. Transform to distance and severity of echo tunnel, trigger anything higher than 10 dB, configurable.
- Constellation display data would be extracted after other issues were found, such as FEC, T3, T4, RxMER issues. For now, just plot and calculate stats of mean, variance, and third moment.

3. Actions

- A CM must be triggered for an action to apply.
- We apply a weight to each trigger type (management information base (MIB), and trigger, so that a given measure could have more than one trigger).
- Score: Create a score for each measure based on the following rules.
  - 1 point for each error or boot issue
  - 1 point for each percent uncorrectable
  - 1 point for each trigger (RxMER, ConDisp), each time
  - 1 point for each trigger in ChEst
- Each day, multiply the score by the weight (score*weight), and sum the results for each CM each day. Use an exponentially weighted moving average (EWMA) scheme to get the CMs their final scores over time. The EWMA approach is a worker that can be exchanged with other approaches.
- Each CM triggered will have its frequency of data collection doubled, configurable for each measurement. Only after having a 0 score*weight for three full days, configurable, will a CM be removed from the action list. The frequency of data collection will be configurable by trigger type, and may cross measures. For example, triggering FEC could result in increasing data collection on FEC by five times the rate, and ConDisp each time FEC is collected, or RxMER at twice the rate.
- All CMs on the action list are prioritized by their score*weight, summed daily over the duration they are on the list (each day's score*weight summed over the contiguous days each CM is on the list, for each CM), or using the EWMA approach as we will for this example application.
- The list is reported for the CMTS, and the CMTS is scored accordingly by summing the measures for the CMs on the list.

4. Modules

The example application that comes with ProOps has several modules, following the structure of ProOps explained previously.

- Control & schedule: A polling scheduler to schedule the data requests in a manner to help the CMTS not get overloaded.
- A data store of results collected, statistics, and scores for each CM, over time. Age off the raw data.
- Translator workers: A translator for each data type pulled.
- Linear calculators: Rapid versions of other calculators or anomaly detectors or other entities which are always applied to data pulled, by data type, before holding in a data store.
- Calculator workers: A calculator or more for each data type pulled.

- Anomaly detector: An anomaly detector for the spectrum and RxMER data. This is just a simple statistical calculator for the first release. A true anomaly detector will be made available later.
- Trigger analyzer: An evaluator to decide on the triggering of a CM, and its weight.
- Severity calculator Worker: A calculator that calculates the severity of the opportunities.
- Cluster Worker: A clustering algorithm that handles work assignments. This is defined for a later release.
- Ranker worker: A calculator and ranker for clusters and CMs together to calculate the measure of severity and rank the opportunities.
- Threshold analyzer sorter worker: A threshold analyzer to indicate dispatchable work, based on rules configured.
- A configuration file to allow changing all the various default values of the application.
- A mechanism (GUI later, via a management interface or the dashboard application) for accessing the list of CMs (and clusters) to act on, and a way to compare their severity.
- Work queue: The queue of work to be performed, organized in a FIFO manner.

The example application is untested in live plant, so the initial configuration described may be far off from a useful combination of settings. The settings will allow verification of function, and be instructive for configuration and developing more functionality, as we now explain.

Configuring an Application

Applications are defined by how the configuration connects the workers. Thus you can define a new application from an existing one by changing the configuration, or adding new workers to the worker environment and configuring their use, or a combination of these actions. Adding a new worker is outside the scope of this technical report, but we will explain how to configure an application using existing workers as it further explains how to make use of ProOps for PNM uses.

ProOps has several base workers that must be configured for an application to work, as opposed to optional workers that can be configured or not depending on need.

Each PNM measurement can be configured on multiple dimensions, aside from the obvious parameters such as network scope for the deployment and other factors controlled by the context of the deployment, or settings for specific PNM measurements like sampling period for FEC statistics. Some of the settings specific to ProOps that should be evaluated for suitability of default settings include the following.

- Time between data requests to a network element for a given data type.
- Time to wait for a time out on a data request.
- Triggering based on range, standard deviation, linear or curve fitting regression parameters, maximum, minimum, mean, median, mode, third or fourth moments, or other user defined statistics including machine learning or artificial intelligence analysis outcomes.
- Weight given to a performance measure as the result of a trigger, such as the range of a measure times a constant factor, or the square of the sum of the uncorrectable errors in the last hour.
- Which worker or driver to use for a given request, and where to send the output of a worker.
- Various queue management settings to control workers assigned to work.

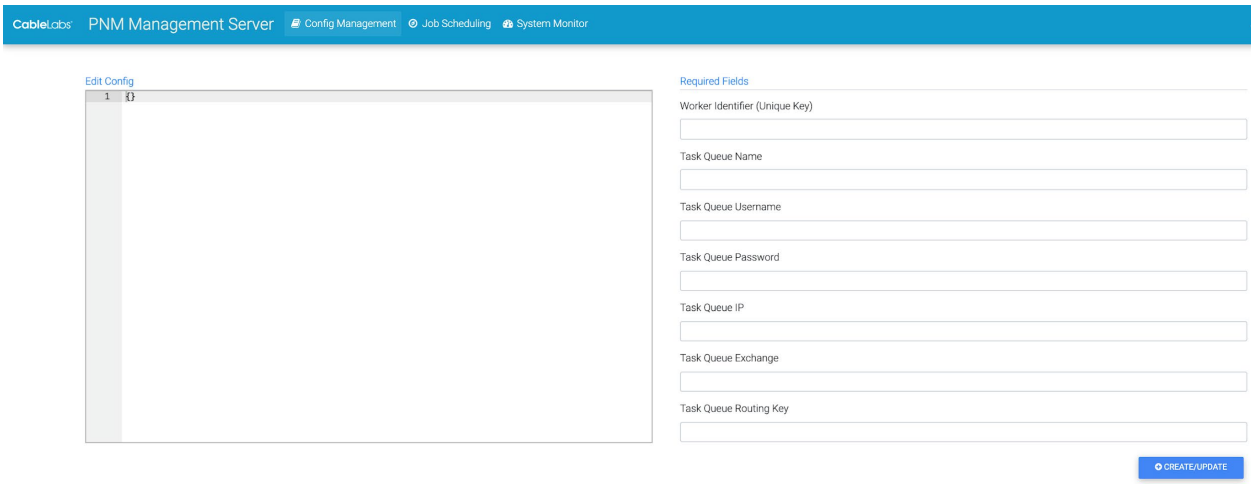

Figure 4 – Configuration management server web GUI index page (task queue configurations).

See Figure 4. On the index page of the configuration server, the task queue information can be configured. The task queue information is essential for all internal communications. When workers start, they request task queue configuration along with other configurations from the configuration server through RESTful APIs.

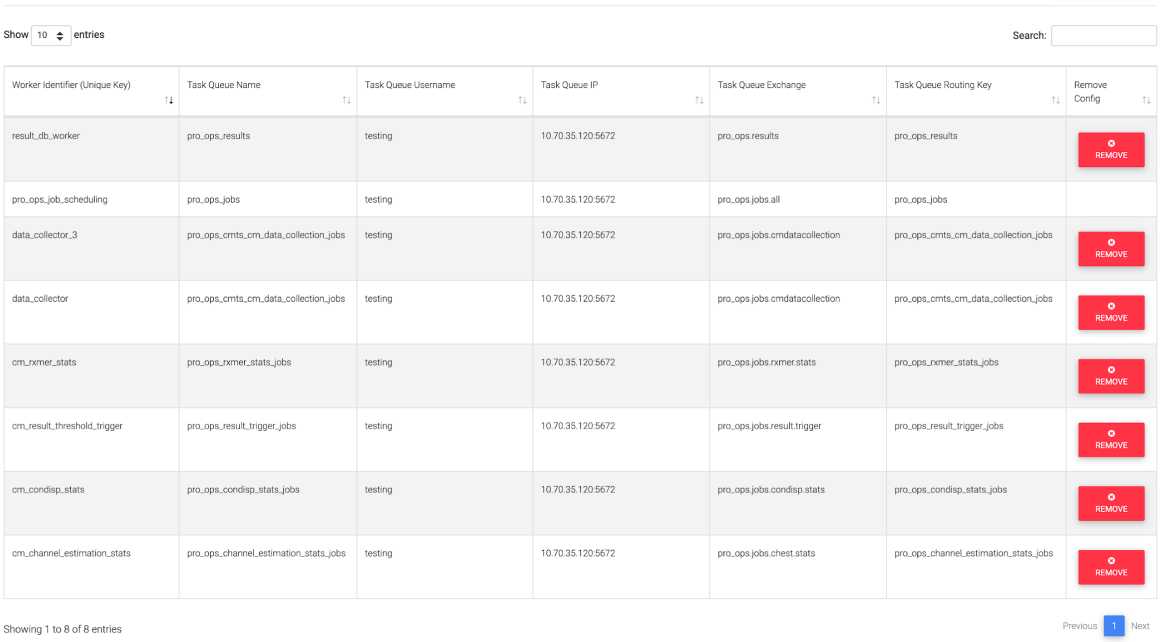

Figure 5 – Configuration management server – defined workers.

Defined workers are listed at the configuration server's index page, shown in Figure 5. When a new worker starts up, the configuration server uses the worker identifier to determine which configuration the worker uses and replies the RESTful API call with the worker's configuration. The worker identifier is a unique key or name that each worker owns. The worker configuration can be removed by clicking the "REMOVE" button.

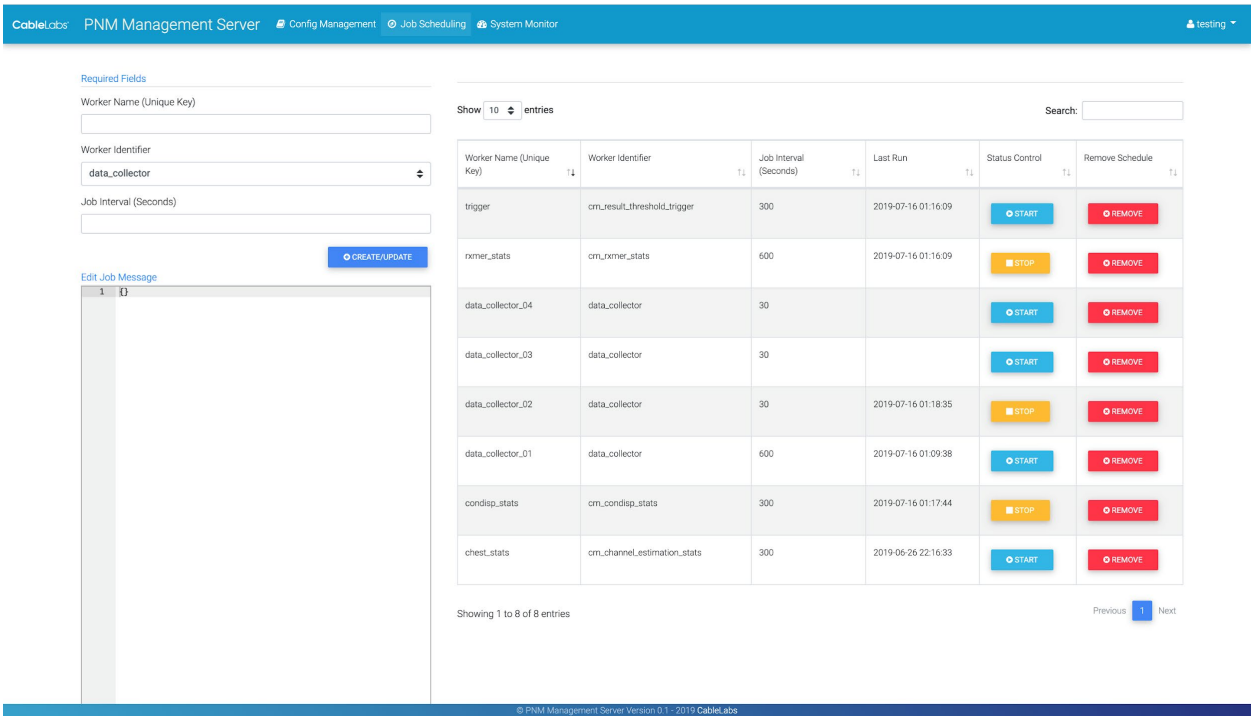

Figure 6 – Job scheduling.

The job scheduling page, shown in Figure 6, displays all defined job schedules. When a new scheduling is started, the configuration server sends a message to the task queue which the job scheduling worker is subscribed to. The job scheduling worker will then add the job to its scheduling queue and send task messages to the task queue periodically based on the job interval.

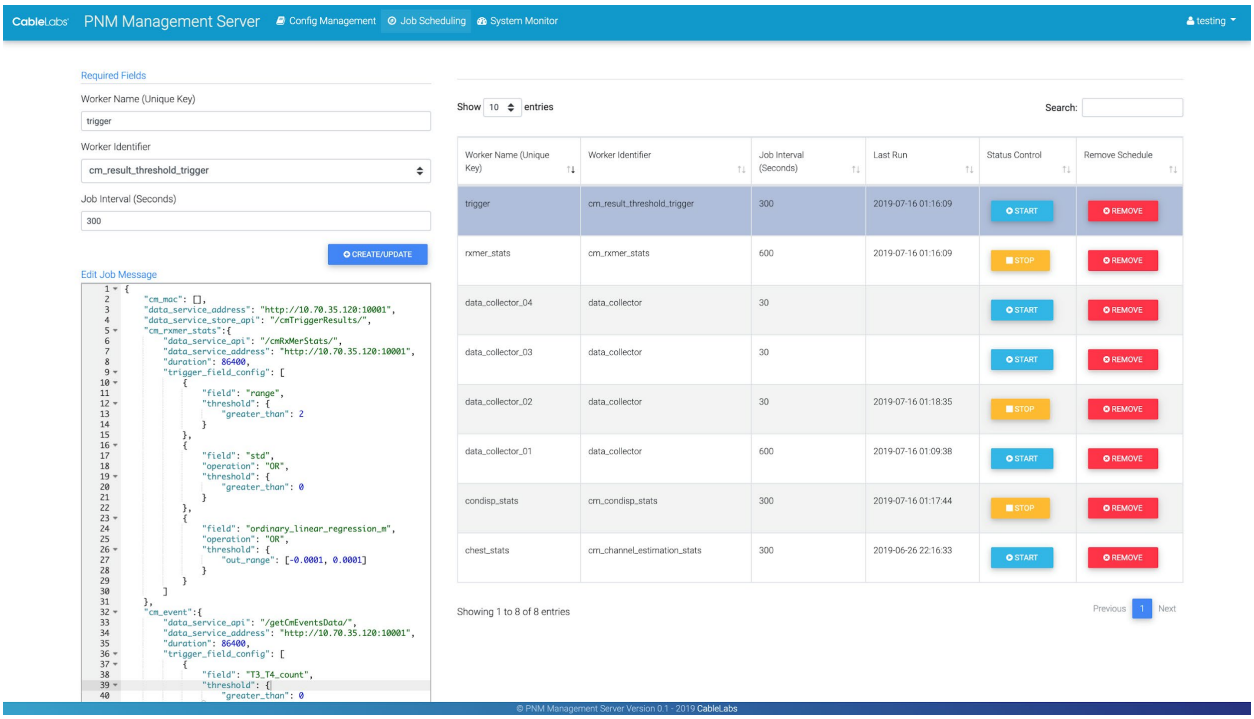

Figure 7 – Configuring job scheduling.

Here in Figure 7 is an example of what can be added to the job scheduling configuration. Other than job interval configuration, we can also add configurations such as data service address and data store API to let the worker know where to retrieve data and where to store results. For example, by configuring the "cm_rxmer_stats" configuration field, the worker will retrieve data from http://10.70.35.120:10001/cmRxmerStats with a backlog duration of 86400 seconds (a day), and find all CMs that have an RxMER range (max - min) greater than 2 dB, or standard deviation greater than 0 (which is set artificially small just for demonstration purposes), or an *m* value (slope in ordinary linear regression) that is smaller than -0.0001 or greater than 0.0001 (this is also set to a very small range as an example). Figure 8 shows an example where we set a greater than 0 threshold for T3 and T4 device event log errors.

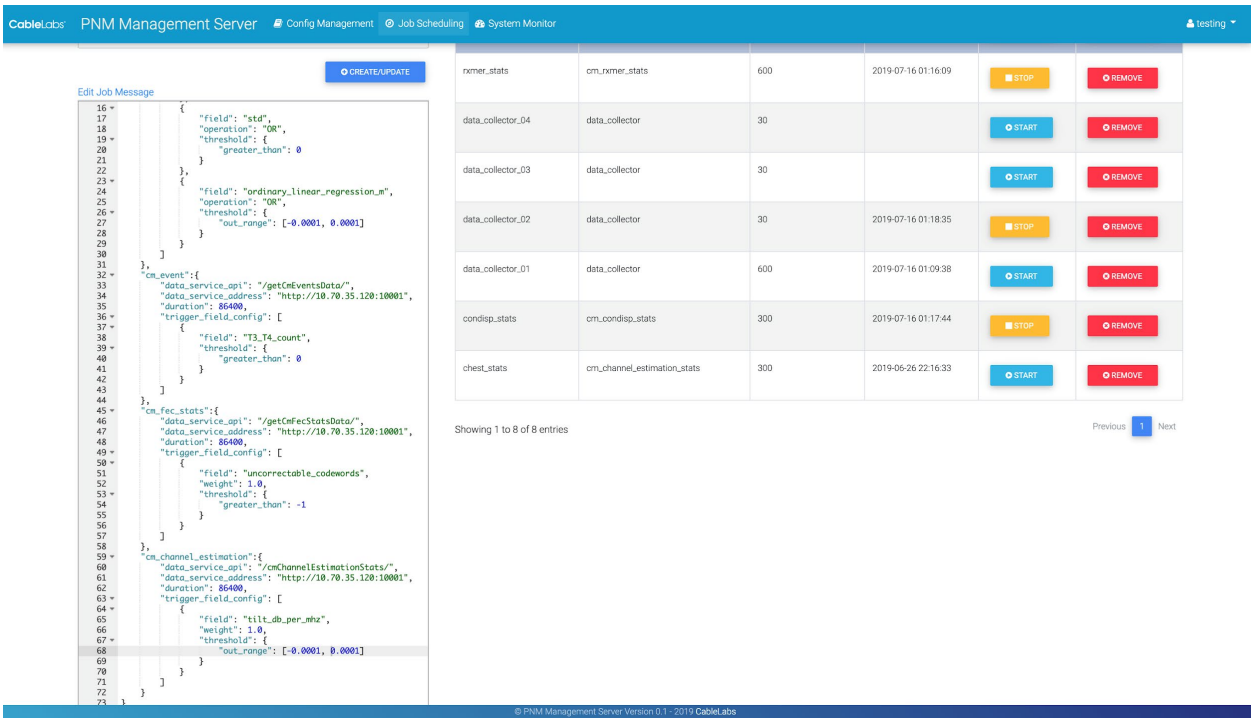

Figure 8 – Configuring for T3 and T4 device event log errors.

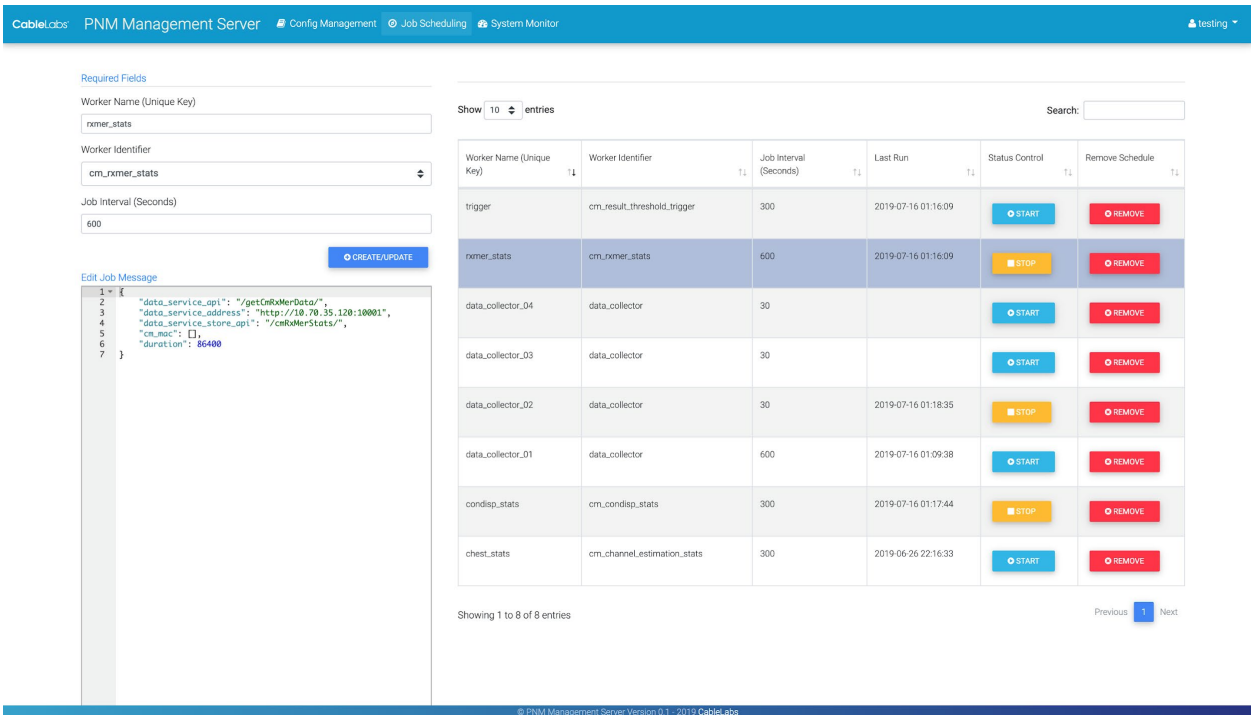

Figure 9 – Configuring the RxMER statistics worker.

The job scheduling of the RxMER statistics worker (see Figure 9) is another example of how to configure a worker. The RxMER statistics worker only requires brief configurations that include job interval, data service API, data service address, duration, and cm_mac list. The job interval defines how frequently the

job is scheduled. The data service API and data service address defines where the RxMER statistics worker retreives RxMER data. Duration defines the data retrieving window from the moment the data is retrieved. The cm_mac list helps filter interesting cable modem MAC addresses. When the list is empty, all CM data will be retrieved; otherwise, only CM RxMER data from cable modems in the cm_mac list will be retrieved. The output from the RxMER statistics worker includes "range", "std", “max”, “min”, “mean”, "ordinary_linear_regression_m", and “ordinary_linear_regression_c”.

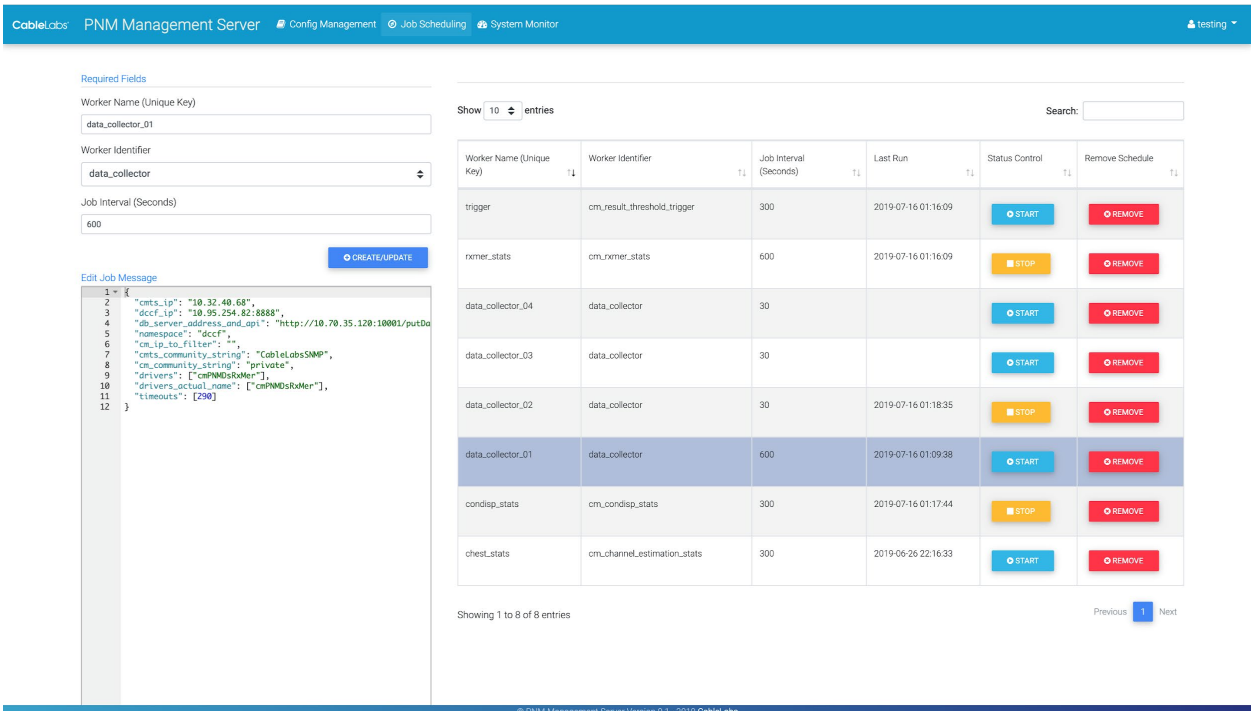

Figure 10 – Configuring a data collection worker.

Shown in Figure 10, the data collection worker is a special worker that interacts with the CCF. In this example, based on the configuration, the data collection worker is configured to run every 600 seconds. It points to a CCF instance with an IP address of 10.95.254.82:8888, and it collects cmPNMDsRxMer data from CMTS 10.32.40.68. The collected data are automatically decoded and stored using the API defined in “db_server_address_and_api”. The cmts_ip field can be a list, and can contain multiple CMTS IP addresses.

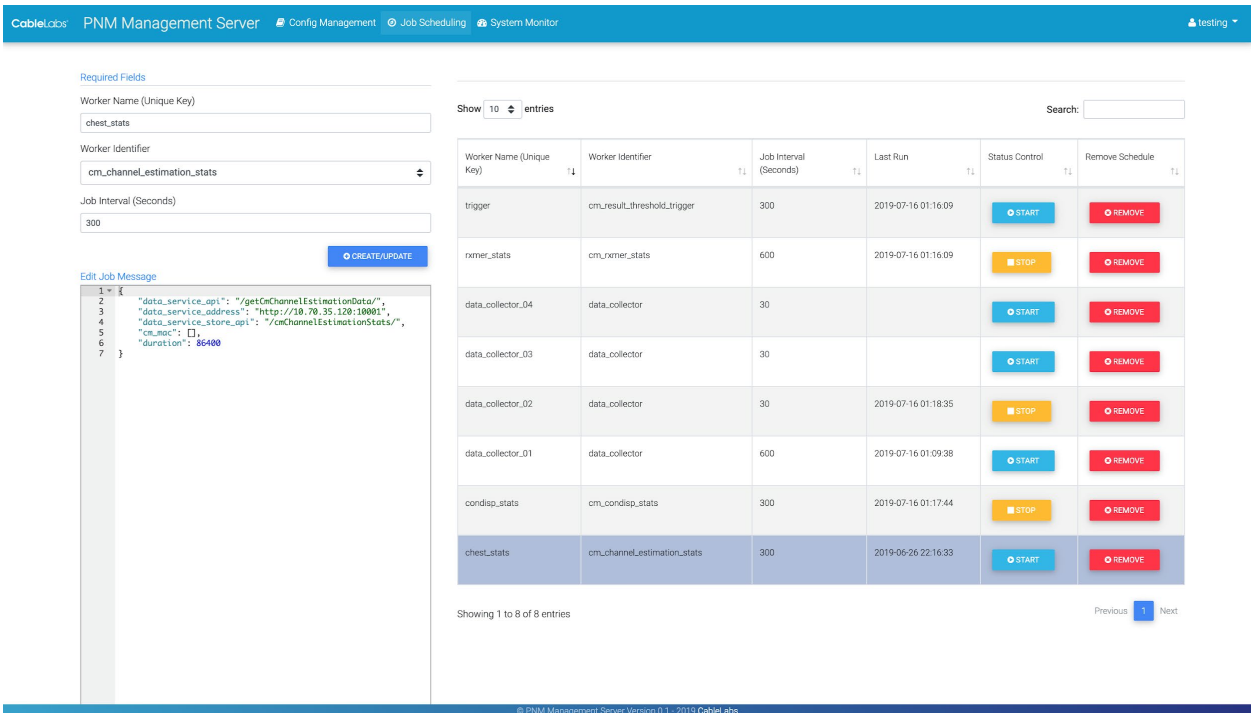

Figure 11 – Channel estimation statistics worker.

Figure 11 shows the CM downstream channel estimation worker, which has a configuration that is similar to CM RxMER statistics worker's configuration. The output includes calculation results on the tilt of the channel estimation and inferences on potential echo distances.

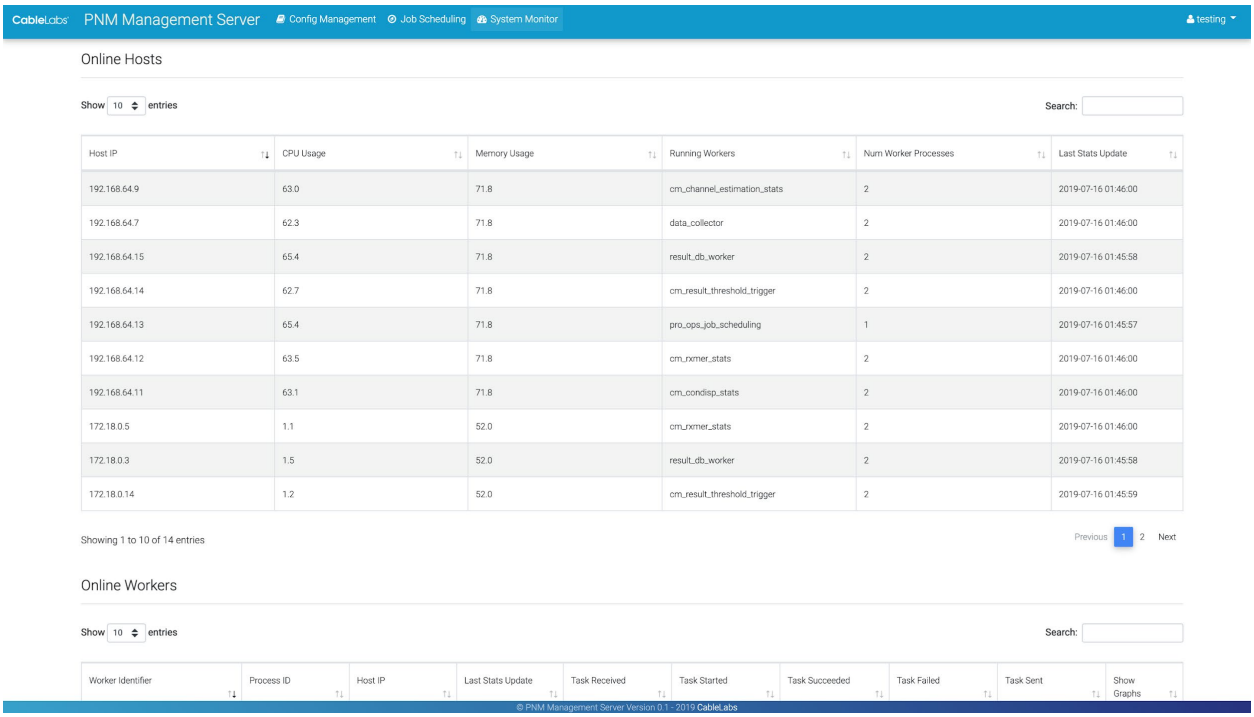

Figure 12 – Top-level system monitoring.

Figure 13 – Detailed worker system monitoring.

Figures 12 and 13 show the system monitoring capabilities that come with ProOps. The configuration server has a page that displays worker running statistics and host resources. In Figure 12, workers are running in separate docker containers and these containers show up as different hosts. The system monitoring page shown in Figure 13 also has a table that displays more detailed worker statistics. The statistics are calculated on a per-worker process basis, and include how many tasks are received, started, succeeded, failed, and sent. As an example, some data collection tasks failed here because of lab equipment changes and wiring changes during collection.

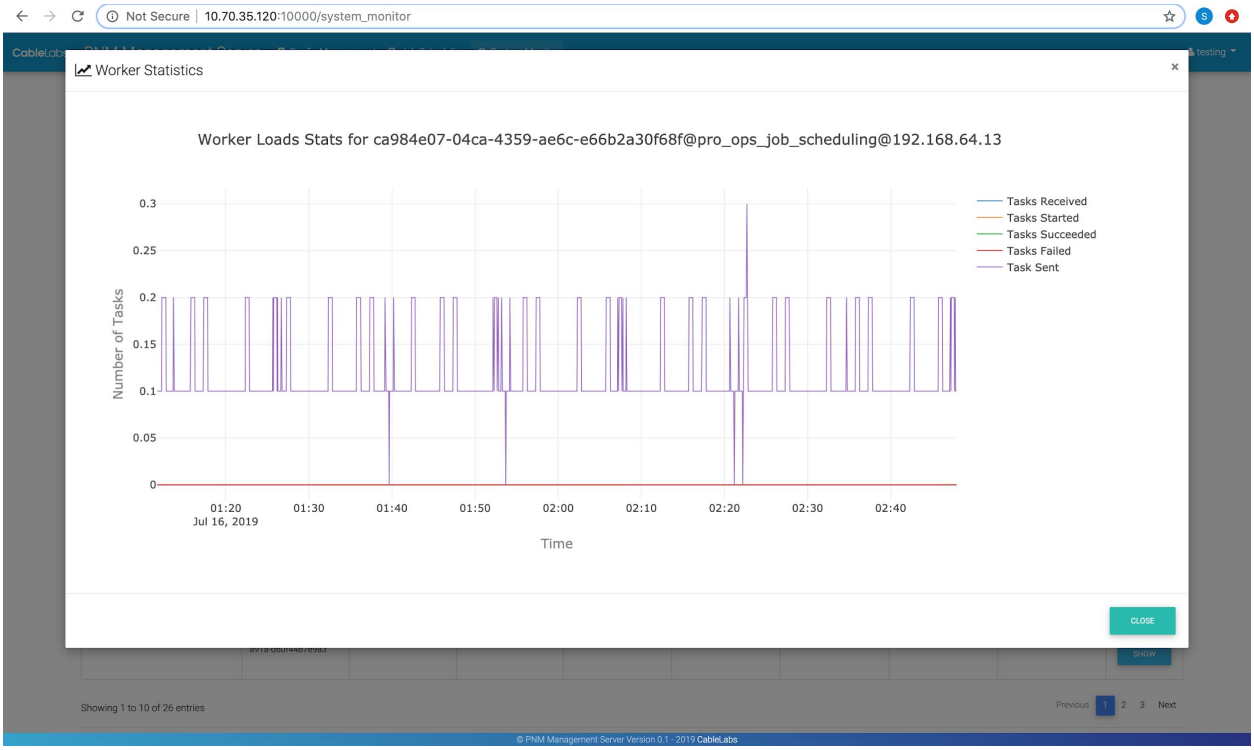

Figure 14 – System monitoring graph.

Here in Figure 14 is an example visualization of how the job scheduling worker is handling tasks every few seconds. The graph shows average numbers for each counter over time. This output provides a quick and easy way to see whether system resources are sufficient, or ProOps is properly configured to perform well, and as intended for the application configured.

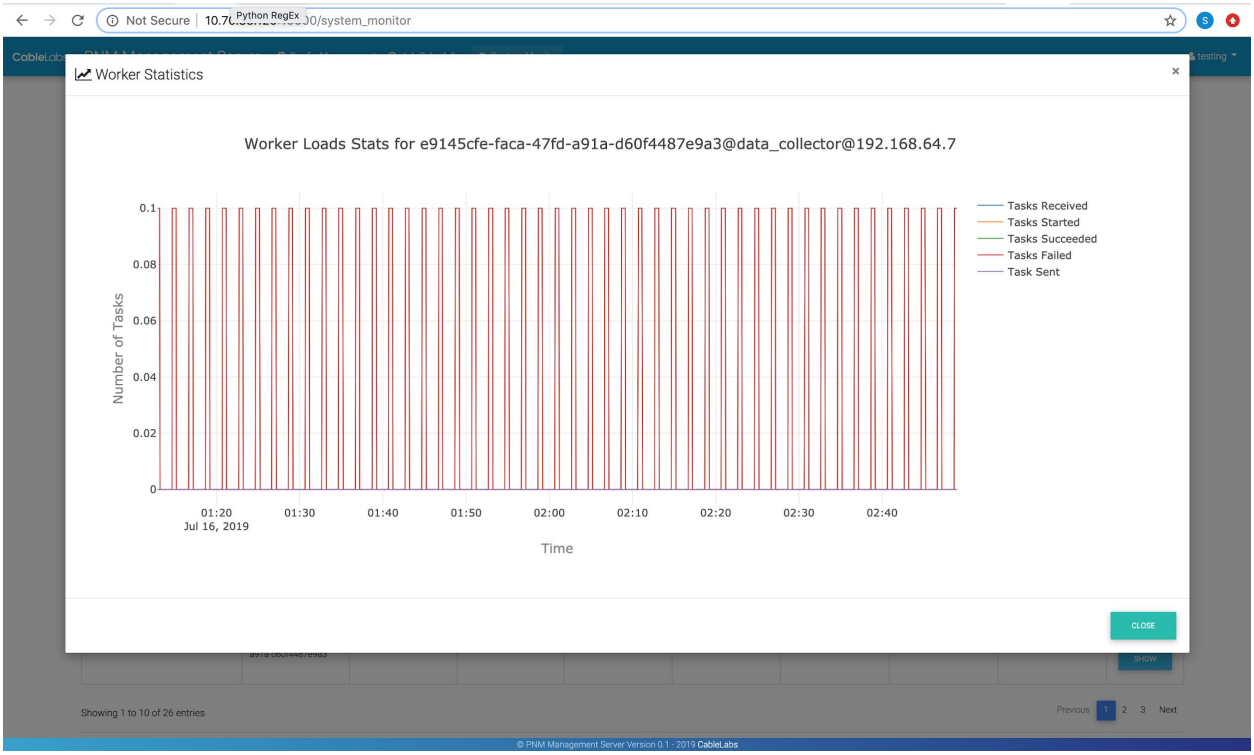

Figure 15 – System monitoring graph of data collection.

See Figure 15. Here is an example visualization of how the data collection worker is handling tasks every few seconds. The graph shows average numbers of each counter over time. This is useful for making sure the data collection resources are not over taxed, or the network is not over taxed as well.

Envisioned Use Cases

We created ProOps to support several use cases for both operators and vendors. Because of this, we had to create it to support several broad advantages.

- Scalability – The architecture of ProOps extends CCF in a modular, layered way. As such, we have full options of ways to package the software, and scale it to any use case, from a desktop deployment to a data center to a cloud architecture.
- Low adoption friction – Existing deployments of CCF can easily be extended to host ProOps. Once ProOps is installed, applications are simple updates to ProOps as additional, optional software modules that link into it. Each additional module installed within ProOps behaves like an update.
- Flexibility and configurability – Because ProOps allows anyone to add workers to the environment and configure rules for their interaction, each deployment of ProOps can be configured independently and in a broad range of ways. That flexibility provides full control to the user, which means it can be configured poorly, but because it is flexible it can be easily corrected and adjusted to fit changing needs too.
- Actionable output – The use of layers of workers as explained previously will allow ProOps to be configured so that the output can be acted on with confidence. If conditions change and the repair

work that it suggests is no longer as desired, then adjustments to the configuration can correct that easily.

- Interworkability – Use of CCF allows one data source to support many applications. Likewise, ProOps is a platform that can support many applications in the same platform, allowing them to share information throughout the process of turning data into action. New applications built in ProOps can run with other existing applications in the same platform deployment. Hardware and virtual machine (VM) resources are the main consideration only.
- Modular – ProOps is modular, so you can replace elements like the worker queue, databases, and worker modules with your own creations or to fit our own corporate information technologies (IT) policies.
- Speed – CableLabs has and will continue to improve the speed of execution of ProOps, but what is important to note is that an application can be created in ProOps rapidly, so it brings speed of execution and deployment together. As we develop more basic workers for the environment, much more will be possible with just a simple reconfiguration of the platform to utilize existing workers in different ways.

Several operator use cases can be supported by ProOps.

- Experiment with PNM – You can connect ProOps to a small test network and test its capabilities against your expectations, and tune it for your own needs. Then depending on the outcome of your experiement, you can take the solution and deploy it in your network in many different ways, whichever suits your situation best.
- Build and test a new PNM idea or solution – As ProOps is a flexible platform, you can create your own PNM method and test it in an example or live network as you see fit, and even tune the platform to your new solution before full deployment.
- Development environment – Because of its flexibility and open architecture, you can develop prototype solutions inside of ProOps, essentially making it your team's development environment for operations solutions.
- Network sampling – ProOps was created to schedule work, so you can schedule network sampling for any operations need, including creating requirements for PNM, or to build a business case for a conceived PNM solution.
- Grow your own PNM program – Because you can use ProOps for the entire chain of PNM solutions, you can experiment, build, and deploy what you come up with, and support it yourself if you wish. Each development in ProOps works with the previous, so you can grow the entire solution and keep it updated as network needs change, keeping your solution up to date, and tailored for specifc problems, architectures, or needs in an area.
- Gather requirements for vendor supported solutions – Because you can use it as an experiment platform or to develop a business case for a PNM effort, it can be used to support a vendor-supported PNM solution too.

Vendors also can use ProOps to their advantage.

- Rapid prototype new potential solutions – Vendors can use ProOps to develop their new PNM products or services, and
- A framework to support products and services – Vendors can use ProOps as a vehicle to deploy their solutions, to ease adoption and reduce the operations impact of the deployment, and to work in harmony with home-grown or other vendor solutions in an integrated manner.

- Free sample solutions – Vendors can deploy their proprietary solutions as example offerings with limited capabilites so operators can test their solution before purchasing the full enterprise supported version.
- Network sampling for developing operator specific solutions – Vendors can deploy a network sampling version of ProOps in an operator's network to help them determine the benefits of implementing a specifc PNM or operations solution, to help get over the business case uncertainties that may hinder purchase decisions.
- Rapid, flexible data collection for consultation – Vendors who work as consultants to operators can use ProOps to collect network data to look for a specific problem, or to support a specific network issue they are trying to resolve.

ProOps was made to turn data into action, so we expect network operations and engineering personnel will discover new ways to make use of it that we have not yet defined.

Conclusion

ProOps is free to use by CableLabs vendor and operator members. Contact the authors of this paper to obtain a copy of the software, and to get help configuring it for your needs. We encourage everyone who can access ProOps to use it in any way they envision, from reviewing the architecture to taking those advantages in their own developments on up to full use and deployment of the code base.

ProOps is a platform constructed for turning data and information into action. It is built around well defined, general steps that facilitate making decisions automatically, but while keeping control fully in the hand of the users. Rather than relying on assumed experts at hand to review network data to determine what needs to be done, ProOps provides an environment to automate that work. This means limited expert resources can be shared through ProOps, thus extending the effectiveness of expertise. For example, CableLabs expects to work with the PNM community to build into ProOps much of what will be documented in the forthcoming DOCSIS® 3.1 PNM Best Practices document.

ProOps is well suited for PNM, but it can be used for turning any data into action, really. With CCF being fully flexible to gather most any network or system data source through creation of a driver, ProOps likewise can collect that data, analyze it, add context, translate results into potential work, then select the work that is most important to do. Network operations efficiency and improving service reliability are the intended goals of ProOps, but only our imaginations will limit what it can do.

We hope, and fully support, operators and vendors contributing code to the C3 repository for sharing solutions, guiding the industry to solve problems, and sharing ideas. CableLabs intends to build workers with new capabilities and share them in ProOps as workers with which members can build soluitons, and for CableLabs to build other proof-of-concept appications. In the months ahead, the power of ProOps will increase due to the contributions from the entire community. We hope capable operators and interested vendors will work with us to develop workers and applications that solve important PNM needs so that both operators and vendors may take advantage from the PNM capabilities that CableLabs provides.

Abbreviations

API	application programming interface
C3	common code collection
CableLabs	Cable Television Laboratories
CCF	common collection framework
CM	cable modem
CMTS	cable modem termination system
CMVA	cable modem validation application
dB	decibel
DOCSIS	Data-Over-Cable Service Interface Specifications
EWMA	exponentially weighted moving average
FEC	forward error correction
FIFO	first in first out
GIS	geographic information system
GUI	graphical user interface
IP	Internet protocol
IPR	intellectual property rights
ISBE	International Society of Broadband Experts
IT	information technologies (information technology)
MAC	media access control
MIB	management information base
NDA	nondisclosure agreement
OODA	observe, orient, decide, and act
PNM	proactive network maintenance
ProOps	Proactive Operations
RxMER	receive modulation error ratio
SCTE	Society of Cable Telecommunications Engineers
TFTP	trivial file transfer protocol
VM	virtual machine

Bibliography & References

[1] https://danford.net/boyd/

[2] *Boyd's OODA Loop and the Infantry Company Commander*, A. Bazin, Infantry Magazine, 2005.

[3] CableLabs Proactive Network Maintenance Combined Common Collection Framework Architecture Technical Report, CL-TR-XCCF-PNM-V01-180814, August 14, 2018, Cable Television Laboratories, Inc.

[4] https://code.cablelabs.com/proactive-operations-platform

[5] https://www.cablelabs.com/cable-network-reliability-proops-platform-for-pnm-and-more

