Detecting the Performance Impact of Upgrades in Large Operational Networks

Ajay Mahimkar[§], Han Hee Song[§], Zihui Ge[‡], Aman Shaikh[‡], Jia Wang[‡], Jennifer Yates[‡], Yin Zhang[§], Joanne Emmons[‡]
The University of Texas at Austin[§] AT&T Labs – Research [‡]
{mahimkar,hhsong,yzhang}@cs.utexas.edu
{gezihui,ashaikh,jiawang,jyates,joanne}@research.att.com

ABSTRACT

Networks continue to change to support new applications, improve reliability and performance and reduce the operational cost. The changes are made to the network in the form of upgrades such as software or hardware upgrades, new network or service features and network configuration changes. It is crucial to monitor the network when upgrades are made because they can have a significant impact on network performance and if not monitored may lead to unexpected consequences in operational networks. This can be achieved manually for a small number of devices, but does not scale to large networks with hundreds or thousands of routers and extremely large number of different upgrades made on a regular basis.

In this paper, we design and implement a novel infrastructure MERCURY for detecting the impact of network upgrades (or triggers) on performance. MERCURY extracts interesting triggers from a large number of network maintenance activities. It then identifies behavior changes in network performance caused by the triggers. It uses statistical rule mining and network configuration to identify commonality across the behavior changes. We systematically evaluate MERCURY using data collected at a large tier-1 ISP network. By comparing to operational practice, we show that MERCURY is able to capture the interesting triggers and behavior changes induced by the triggers. In some cases, MERCURY also discovers previously unknown network behaviors demonstrating the effectiveness in identifying network conditions flying under the radar.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management*

General Terms

Management, Performance, Reliability

Keywords

Network Upgrades, Performance Impact, Change Detection, Statistical Data Mining

1. INTRODUCTION

IP networks have become the unified platform that support a rich and extremely diverse set of network applications and services, including traditional IP data service, Voice over IP (VoIP), smart mobile devices (e.g., iPhone), Internet television (IPTV) and online gaming. Network performance and reliability are critical issues in today's operational networks because many applications place increasingly stringent reliability and performance requirements. Even the smallest network performance degradation could cause significant customer distress. In addition, new network and service features (e.g., MPLS fast re-route capabilities) are continually rolled out across the network to support new applications, improve network performance, and reduce the operational cost.

Network operators are challenged with ensuring that network reliability and performance is improved over time even in the face of constant changes, network and service upgrades. Network upgrades themselves can result in significant changes in network or service performance. Some of these changes may be as designed - for example, a software upgrade could be introduced to a network specifically to address performance issues, such as higher than desired router CPU utilization. However, other performance changes resulting from network upgrades may be completely unexpected, and even highly undesirable with negative impacts on customers. More importantly, such network behavior changes cannot be fully reproduced or predicted in a lab testing environment. Therefore, it is important to monitor network behavior and understand the impact of network upgrades on service performance.

Network operators continuously monitor the network so that they can rapidly detect and recover from outages and performance impairments. Tremendous effort is devoted to the real-time detection of major issues, such as fiber cuts or congestion, so that they can be rapidly repaired. However, an operator that purely focuses on each individual event in isolation may fail to identify critical patterns which may be indicative of a larger issue. Carefully managing network performance thus also requires examining network events holistically - exploring series of events - large and small - instead of purely focusing on each large event in isolation. The end goal is to identify actions that can be taken to improve overall network performance and reliability. Such actions can take many forms, including software bug fixes, hardware re-designs, process changes and/or technology changes.

Key Performance Indicators (KPIs) are used to track longer-term service and network performance. KPIs cover customer-perceived performance (e.g., network packet loss, service glitches) and network performance (e.g., protocol flaps, line card crashes). KPIs also track network element health (e.g., router CPU, memory).

In this paper, we focus on network upgrades and their impact on a wide range of KPIs. We collectively refer to the set of different network upgrades that could potentially cause network behavioral or performance changes as *triggers*. We define *network upgrades* to be significant network changes designed to enhance the network or service performance. These network upgrades include router software or hardware upgrades, new network or services features, and other fundamental network configuration and policy changes designed to impact how the network or service operates across customers. It is important to distinguish network upgrades from individual customer provisioning which is a continual and extremely frequent activity in a large ISP network. These provisioning activities only impact individual customers and generally do not have a network wide impact.

The current operational practice in determining the impact of a network upgrade on a set of KPIs involves manual comparison and tracking of each KPI before and after the upgrade at each network location. This process is clearly cumbersome and tedious, and is simply impossible to scale to large numbers of network nodes and KPIs. The result is that critical issues fly under the radar, impacting customers and causing unnecessary distress. Thus, innovative and automated solutions are required to effectively analyze network behaviors across a broad class of KPIs and across all network nodes.

Challenges. In this paper, we take an important step towards building an automated infrastructure for understanding the impact of network upgrades on network performance. However, achieving this required addressing several challenges, including the following.

1. **How should we identify network upgrades?** Operational IP networks are continuously undergoing network maintenance activities, including network upgrades, provisioning of new customers, replacement of failed network components and so on. Such activities leave traces in network logs - for example, workflow logs capture all commands typed on routers. However, identifying which activities are network upgrades versus other activities is extremely challenging. Operations personnel manually track many types of network upgrades, but this information is distributed across large numbers of individuals and even work groups. It is thus very challenging to capture. And given that this information is manually collated, it is inherently error prone and incomplete. We thus desire more automated ways of capturing and identifying such fundamental network upgrades, extracted from the wealth of network-related data.
2. **How can we determine the impact of network upgrades?** We are focusing in this paper on *persistent* changes in network performance that result from network upgrades. We must thus devise a metric that detects such persistent performance changes and distinguishes them from short-term or transient performance fluctuations.
3. **What are the commonalities among network elements with similar detected behavior changes?** When a set of routers all demonstrate a similar change in network performance with a given network upgrade, it becomes desirable to identify what this particular set of routers have in common that distinguishes them from other routers which did not experience the same performance changes. This can be an important step in further revealing insights that may help in explaining the behavior changes. For example, do the routers that exhibited the change all have a common operating system or are a common model? Or does the change only happen to routers with certain numbers or types of customers? There are a tremendous number of different potential characteristics that could be relevant here, which makes such an analysis extremely challenging.
4. **How can we detect network wide aggregated behavior changes?** It may be impossible to identify statistically significant changes in time-series that are very sparse such as rare events on a single router. However, if we aggregate events across larger numbers of routers, then we may observe significant statistical changes. Thus, it is critical to examine aggregated time-series across multiple network elements to ensure that we capture all performance changes. But at what aggregation level should such an analysis be performed? Should we look across the whole network or just across a given router type or router role? If a given change in behavior impacts only a specific class of routers, and if we look at a much larger aggregation (e.g., network wide), then we may lose this effect in the larger set of events. Thus, it is crucial to identify the appropriate aggregation level for detecting behavior changes.

Our Approach and Contributions. We design and implement *MERCURY*, an automated tool for monitoring and determining the performance impact of network upgrades. MERCURY systematically extracts interesting triggers to identify network upgrades activities. In the current implementation, we identify the triggers from router configuration files and workflow logs, as these capture configuration-related changes. MERCURY then monitors a wide range of network key performance indicators (or KPIs) and detects persistent behavior changes that temporally correlate with our network upgrades (or triggers). In particular, MERCURY addresses the above challenges as follows:

1. MERCURY identifies the interesting triggers from a large set of maintenance activities by capturing two key metrics: (i) *rareness* of the activity and (ii) *coverage* or *skewness* of the activity. The rationale behind it is that upgrades are typically rare events and they tend to cover multiple network elements. By contrast, other types of maintenance activities such as provisioning and network diagnosis are much more frequent and likely to be performed on a single network element. By using the rareness and skewness metrics, MERCURY successfully filters out the majority of non-upgrade related maintenance activities.
2. To detect persistent behavior changes and distinguish from transient changes, MERCURY applies a non-parametric rank-based behavior change detector. For each KPI of interest, MERCURY outputs multiple change-points, each of which has a score indicating the significance of the change.
3. To effectively reduce the computational complexity involved in identifying common attributes among the network elements that have consistent behavior changes, MERCURY first associates triggers and change points, and filters out triggers that have no performance impact. Then, MERCURY applies *statistical rule mining* techniques to identify common attributes among all network elements that experience consistent behavior changes for a given trigger.
4. To detect aggregated behavior changes, MERCURY first time-aligns KPI event series by the trigger timestamp and aggregates them across multiple locations where no behavior change is detected at individual location, and then applies change detection on the aggregated event-series. The commonality rules serve as a guideline for the appropriate aggregation levels.

We systematically evaluate MERCURY using data collected from a large tier-1 ISP network. In our current prototype, we gather KPI statistics from SNMP measurements (router CPU and memory utilizations) and router syslogs. Network upgrades (triggers) are identified from configuration files and workflow logs. The first step of our evaluation analyzes the effectiveness of our automatic detection of triggers. We demonstrate that MERCURY is able to achieve low false positives and false negatives when compared to hand-labeled triggers that network operations deem important. We also identified a few new, but important triggers that were previously unknown to network operations. After identifying the triggers, we analyze the network behavior changes detected by MERCURY. This analysis confirmed some of the Operations team's earlier findings as well as revealed previously unknown network behaviors.

Paper Organization. The rest of the paper is organized as follows. We provide a taxonomy of network upgrades and behavior changes in Section 2. We describe the design of MERCURY in Section 3 including detailed algorithms for trigger identification, behavior change detection, association of triggers and change-points and aggregate change detection. We present the evaluation results in Section 4. We share our deployment experiences with interesting case study findings in Section 5. Finally, we review related work in Section 6 and conclude with discussions for future work in Section 7.

2. NETWORK UPGRADES AND IMPACT

In this section, we give a detailed description of network upgrades and key performance indicators (KPIs).

2.1 Network Upgrades

In today's operational IP networks, the majority of network upgrades are planned and proactively performed, although others may be reactive and can be the result of external network conditions such as Denial of Service (DoS) attacks and worm propagations.

We expect that the majority of network upgrades are implemented via configuration changes or changes in the software deployed within the network routers. We do not consider triggers external to the service provider network such as a change in a peer network because they may not be directly observable. Network operators perform software upgrades and configuration changes via the command line interface on routers. We thus focus on triggers observable via operating system upgrades, firmware upgrades (RAM or ROM versions) and significant configuration changes.

Operating System and Firmware Upgrades. Operating system and firmware upgrades are typically made to introduce new features to the network (e.g., fast re-route to improve re-convergence times) or to eliminate bugs in older versions. Such upgrades are made on a time-scale of months.

Configuration Changes. Configuration changes are made by the network operators using specific commands on the routers implemented via the command line interface. The changes can either be applied to each interface on the router, across multiple interfaces or across the whole router.

Generally, the triggers are made one network element at a time and then applied across multiple elements. For example, an operator upgrades the operating system on one router, and then applies the changes to all routers of certain types. Extensive testing is performed on router software and hardware before new software or hardware is deployed, or before new types of configuration changes are made in an operational network. The goal of this testing is to attempt to prevent bugs and poorly performing hardware and software from reaching the network. And such testing is extremely successful in doing this. But despite all best efforts, lab testing is simply unable to replicate the immense scale and complexity of an operational network, and thus there is always the risk that issues may creep into the network. Thus, as changes are rolled out across a network, network operators carefully monitor their impact on network performance. Should unexpected and undesired performance changes or issues be observed, the operators are responsible for driving them out of the network. In situations where the issues are within the router software, repair necessarily involves the router vendor. Extensive lab reproduction and careful software analysis is used to support the diagnosis of the issue; permanent repair will likely involve fixing the relevant software bugs and deploying the resulting new software network-wide.

2.2 Key Performance Indicators (KPIs)

In today's networks, operations teams carefully monitor a set of pre-defined KPIs such as router CPU utilizations, packet loss, delay and routing protocol flaps. The goal is to monitor the effect of the network upgrades on these KPIs. Any behavior change in the KPI induced by the network upgrades is a good indicator to either confirm the desired impact or discover any new unexpected impacts.

We define a behavior change to be a persistent change in performance induced by a network trigger. It may be indicative of either a new problem (e.g., an OSPF configuration change causing behavior change in the number of LSAs) or a fix to an existing problem (e.g., an operating system upgrade to fix router bugs).

In this paper, we focus on behavior changes observed in individual performance event-series. Some changes are instantaneous and result immediately after the trigger (immediate *level-shift*), while others change gradually over time (slow *ramp-up*). Behavior changes can also correlate across multiple locations. These are typically induced by the same triggers applied across different routers. For example, layer-1 timer changes configured across routers should result in improved network-wide convergence times.

Performance Metrics. In this paper, we focus MERCURY on two key data sources - router SNMP measurements and router syslogs. Through these two data sources, we are able to capture an immense range of potential KPIs, providing a solid set of indicators of network performance and health. In fact, by automatically mining *all* of the potential types of time-series that can be captured from these two data sources, we can capture a set of measurements which go well beyond the traditional KPIs used via manual processes today.

SNMP MIBs capture measurements of various parameters on the router, including counts of packets and bytes transmitted through router interfaces, packet errors, and CPU load and memory utilizations. External pollers collect values from the SNMP MIBs at regular intervals (e.g., every 5 minutes), providing a regular stream of average measurements that can be used in real-time or stored historical for later analysis.

In contrast with SNMP measurements, which provide average statistics collected from the router, *router syslogs* are logs written by routers in response to specific events on the routers. Router syslogs capture a very diverse range of events, including protocol and link state changes (up/down), error conditions, warning messages (e.g., denoting when customers send more routes than the router is configured to allow) and even information about environment conditions (e.g., high temperatures). Syslog messages are basically free-form text, with some structure (e.g., indicating date/time of event, location and event priority). This makes them somewhat challenging to automatically analyze, but the immense diversity in the range of different conditions that they capture make these logs extremely important.

There is a large number of network performance event-series one needs to mine in order to detect behavior changes and correlate them with network triggers. Identifying commonality in behavior changes across multiple locations is important to effectively drill-down into results and identifying the root-cause as well as location where the change is prevalent.

3. MERCURY DESIGN

Recall that our mission is to systematically identify any network behavior changes in response to upgrades performed in the network. To achieve this, we decompose the mission into three components as follows.

First, we need a data-driven approach that can reliably infer that any interesting (i.e., potentially impact-bearing) upgrades have taken place in the network – let it be a router OS upgrade, or a configuration change activating a new feature. We choose not to rely on unreliable human communications for MERCURY to learn of such an operation activity – hard to automate the process being one concern, and the information quality being another. In a large ISP net-

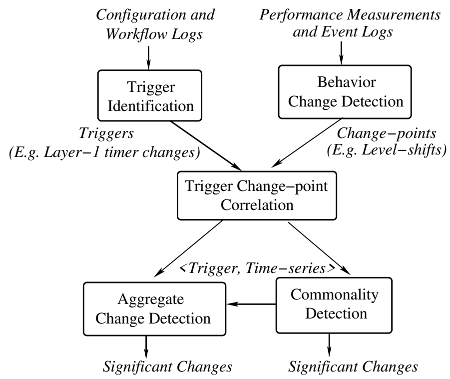

Figure 1: MERCURY architecture.

work, it is simply unrealistic to assume that any one person or one team has the knowledge of all planned upgrades. Moreover, when and where the actual upgrade takes place in the network can often deviate from the planned one. Thus, it is critical to automatically discover the time and location of interesting upgrades, the *triggers*, from the network data (e.g., network configuration snapshots and workflow logs). Note that this does not exclude MERCURY to use external information about triggers, had it been available and reliable.

Second, we need to detect network behavior change as demonstrated by different performance metrics, and associate them with the triggers. There are many statistical techniques that can be used for change detection. We have explored several and included one that is the most effective in MERCURY. However, the challenge here is to construct the time-series of performance metrics so that they produce meaningful result indicating network behavior changes. For example, we need to properly handle rare but significant outliers so that they do not dominate the overall trend of the time-series and bias our analysis.

Finally, we need to make sense of the identified triggers and their induced behavior changes. Here, we want to minimize the impact of false correlation by generalizing the observation of trigger-change relations at individual routers to a set of meaningful high-level rules, which will be reported to network operators for further investigation. Domain knowledge about the type, role, and other attributes or characteristics of routers are incorporated in the generalization process.

3.1 Overview

We now present the system architecture of MERCURY. As shown in Fig. 1, MERCURY takes two types of inputs: (i) configuration and workflow logs that contain information to identify triggers and (ii) performance measurements and event logs to monitor network health/behavior. They are fed into the trigger identification (Section 3.2) and behavior change detection (Section 3.3) modules respectively. The triggers detected are compared against the behavior changes both in time proximity and in the network device affected (e.g., router, line card) (Section 3.4). A list of potential trigger-changes relationship at each individual device is further aggregated and filtered through a commonality detection system (Section 3.5) to become high-level rules for operators to investigate, with each rule presenting "what trigger induced what change(s) on which group of devices". Furthermore, for cases in which the changes in the time-series at individual device level are too subtle to be detected, we also devise a new technique that first time-shifts the time-series to be trigger-aligned and conduct behavior change detection on the aggregate time-series (Section 3.6). Additional cases are opened for network operators to investigate based on output of this module. We next present our design of each module in MERCURY in details.

3.2 Trigger Identification

Our objective in this step is to automatically infer the upgrades performed in the network. As discussed in Section 2, there are two channels that network operators can exercise control over the network – (i) through the router Operating System and device firmware upgrade, and (ii) through changing the configuration of the routers. It is sufficient to focus on identifying network upgrades from tracking operators' activities in these channels.

3.2.1 Identifying Router OS and Device Firmware Upgrades

It is very important to keep a close observation on the network behavior after router OS upgrades, both to confirm the expected behavior change (e.g., improvement in route convergence times), and to uncover any unknown and undesirable network behavior or performance problems induced. The unexpected change can be due to software bugs in the new OS, incompatible configurations, or bad protocol interactions, which did not meet the condition to manifest themselves in the pre-deployment lab testings.

As discussed earlier, we hope to develop a systematic approach in learning "what routers have OS or line card firmware upgraded on what date". To do so, we utilize two data feeds from the network auditing organization – router configuration snapshot and router diagnostic information snapshot. The "snapshots" refer to the data acquired through taking a dump of the running configuration and diagnostic information at the routers. On Cisco routers, these are achieved through issuing commands *show running-config* and *show diag* to the router command line interface (CLI). Included in the router configuration snapshot are the router OS version along with detailed configurations (about routing protocols, CoS, QoS, ACL, etc.), and included in the diagnostic information snapshot are the firmware versions of each line cards along with information such as hardware type, memory size, module status, etc., at the time of the snapshot. In the ISP network, both snapshots are taken from all operating routers in the network on a daily basis.

Our solution to automatic router OS and device firmware upgrade detection is to extract the relevant information from the snapshots and detect any differences across days. In this way, we acquire the upgrade information with the time precision of one day.

3.2.2 Identifying Upgrade-related Configuration Changes

While the router OS and firmware changes always belong to the upgrade activities that need to be closely monitored, router configuration changes however are mainly unrelated to network upgrades. Individual customer provisioning is a continual and extremely frequent activity in a large ISP network and accounts for a majority fraction of the router configuration changes. These configuration changes are unrelated to the network upgrades. Thus, identifying the fundamental configuration changes that are designed to impact (improve) how network operates is a challenging problem.

The data feed that we utilize to track router configuration changes is the *workflow logs*. This is collected from the router access authentication and control system, such as TACACS [12] servers, in which all control sessions to routers in the network are authenticated and all commands issued to the routers are authorized and logged. Each entry in the workflow logs contains the timestamp, the command executed, and the network operators' username and the terminal information. All control activities on the routers including upgrade related configuration change, individual customer provisionings, and testing and diagnosing commands are recorded in workflow logs. We need to automatically identify the network upgrade activities from those.

To filter out the testing and diagnosing command, one natural approach is to acquire vendor product documentation and construct a list of router commands (e.g., *ping* or *show diag*) versus configurations. However, this requires timely update of the list when router OS upgrades. Thus, we take an alternative approach that does not depend on domain knowledge updates. We compare the workflow logs with the configuration snapshots: if a workflow line or its negation (e.g., in Cisco IOS, preceding a configuration with *no* removes the effect of an existing configuration) does not appear in the configuration snapshots, then it does not have lasting impact on the router and can be safely filtered.

Separating network upgrade associated configurations and individual provisioning related ones is much more challenging. One limitation is that the workflow logs have a flat structure – each command line is a separate entry. Yet, the intention of a configuration change can be inferred only through examining the series of configuration commands. Thus, we need to first reconstruct the *configuration session*, the sequence of configuration change lines toward a coherent high-level semantic. Fortunately, network operators and automated configuration management tools usually perform one task at a time, and typically leave sufficient gap between tasks to allow proper verification of the effect of the changes just made. Based on this observation, we group workflow entries from the same user ID and close in time into one session, and apply a threshold of 10 minutes to separate sessions. We find our choice of threshold works sufficiently well in the ISP data.

In order to identify the upgrade related configuration sessions automatically and with minimum dependency on domain expert input, which is unreliable and easily outdated by new router OS version or new features introduced, we develop heuristics that look for configurations sessions that are "out of the ordinary" in some fashion. They may occur very infrequently or they may have an unusual structure in the command sequence. We capture these two properties by looking for rare and skewed configuration sessions.

Rare configuration sessions. Intuitively, if a configuration command is rarely used over a long time (e.g., several months), it is unlikely to be individual customer provisioning activities (as they are frequent) and highly likely to be relating to a network upgrade (e.g., activating a new feature in the network). Yet two challenges remain to utilize this simple idea. Firstly, the same type of configuration change may appear differently in workflow logs due to the different parameters used. For example, the command to set up BGP sessions at different routers includes different peer IP addresses, appearing as different command line in workflow logs. To tackle this, we de-parameterize the commands by removing all IP addresses, device names, network masks, and other numbers. Secondly, not all configuration commands are supported across router types or versions – a command can be mistakenly considered as rare simply because most routers in the network do not support this command. To deal with this, we normalize the frequency count of the different commands by the number of routers on which this command has ever appeared during the time window.

Our test for rareness is then simply comparing the normalized frequency count of de-parameterized configuration command to a threshold (e.g., 4 per router in a six-month window). And the configuration sessions that include any rare command are identified as rare sessions. We will demonstrate the trade-off in the selection of the rareness threshold in Section 4.3.

Skewed configuration sessions. The idea here is to look for the structural difference between provisioning configurations and upgrade related configurations. Individual customer provisioning typically does not repeat the configuration lines, while upgrades may involve applying certain configuration lines to different line cards, different customers, different protocol sessions, etc. For example, network rolling out a new value for the *carrier-delay* trigger timer need to apply the configuration changes to many networking interfaces on each router. The corresponding configuration session becomes skewed in terms of the frequency count of the different commands in the session.

We use the following skewness test: we compute the frequency count, c_i, of different configuration commands in each session. Let μ and σ be the mean and standard deviation of c_i. If the highest frequency count is greater than $\mu + 6\sigma$, then the session is identified as a skewed configuration session. Our approach is consistent with the heavy hitter detection [7, 34, 41] in the statistics literature.

3.3 Behavior Change Detection

Given the list of interesting triggers, the next step is to identify behavior changes in network performance event-series. One approach is to compare event-series statistics (*e.g.*, mean, median, or the entire distribution) before and after the trigger over certain time interval. A change score can then be used to help quantify the magnitude of the changes. However, there are two problems with this approach (which we have experienced in practice): (i) Due to overlapping impact scope of network triggers, choosing the right time interval is non-trivial and an incorrect choice often leads to wrong conclusions. (ii) It does not scale with the large number of performance event-series and the number of triggers. If T is the number of triggers, P is the number of performance event-series and L is the number of spatial locations, then the time complexity is $O(TPL)$, which is very high since T, P and L are each on the order of hundreds to thousands.

To address the above shortcomings, we first identify significant behavior changes in the performance event-series and associate them with the network triggers to identify the trigger/change-point pairs of interest. The trigger and change-points form a pair when they co-occur in time and share the same location. In this way, the number of change detection tests that need to be performed is on the order of $O(PL)$, which is two to three orders of magnitude reduction over $O(TPL)$. Also, by associating the change-points with triggers, we eliminate the triggers that do not cause behavior changes. Next, we describe the event-series normalization and creation followed by the statistical test for detecting behavior changes in performance event-series.

Event-series normalization and creation. For each key performance indicator (KPI), we create an event-series by dividing the original KPI time-series into n equal time-bins. For some KPIs (*e.g.*, certain syslog messages), it is desirable to first normalize before creating the event-series. An example of normalization is grouping an event-type such as BGP hold timer expiry across all interfaces on the router. Our experience suggests that the event counts for different interfaces often have a skewed distribution. To prevent heavy-hitter interfaces (*i.e.*, interfaces with significantly more events than the others) from dominating the change detection analysis, we propose to distinguish heavy hitters from non-heavy hitters (using the heavy hitter test described in Section 3.2). We then detect behavior changes in event-series aggregated across non-heavy hitter interfaces as well as those aggregated across all the interfaces.

Change-point detection using rank-based CUSUM. We have implemented and experimented with a number of statistical methods for detecting changes in a given time-series, including changes in means, medians or even entire distributions. Our experience suggests that detecting changes using the raw time-series has the benefit of capturing the changes in magnitude, however they are not robust to outliers and can have high false alarms because of just a single spike in the time-series. Since our goal is to detect persistent behavior changes and eliminate transient spikes, we propose to detect changes on the ranks of the time-series. We use a rank-based non-parametric statistical test CUSUM [33], that detects change-points (level-shifts as well as ramp-ups) in three steps: first identifying a candidate change-point, then applying a statistical test to determine its significance and recursively applying CUSUM to identify multiple change-points. Since the test is based on ranks, it is robust to outliers and requires no special distributional assumptions.

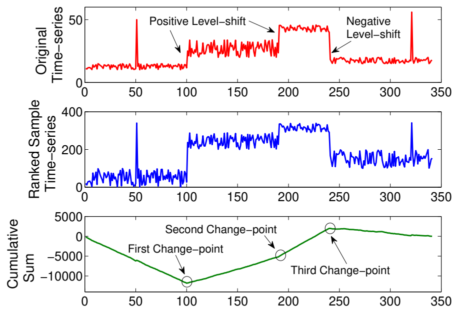

Figure 2: CUSUM example to detect multiple change-points.

1. *Candidate change-point detection:* Let $x_1, x_2, ..., x_n$ be the n samples in an event-series. We rank the samples in increasing order and construct the rank r_i for each sample x_i. In case of ties, we assign average rank to each sample. The cumulative sums are computed as:

$$S_i = S_{i-1} + (r_i - \overline{r}) \quad (1)$$

$S_0 = 0$ and $\overline{r}$ is the mean across all ranks r_i. If the ranks are randomly distributed, then there is no change-point. However, if there is indeed a change-point in the event-series, then higher ranks should dominate in either the earlier or later part of the event-series. Thus, if the event-series contains a change-point after which the values are greater than before (*i.e.*a positive level-shift or ramp-up), then S_i will decrease to a minimum before increasing to zero at $i = n$. The change-point (for positive shift) is the minimizing index. Similarly, for negative shifts, the change-point is the maximizing index. The change score is $S_{diff} = max(S_i) - min(S_i)$.

2. *Significance testing:* Bootstrap analysis is performed to identify if the change is statistically significant. The idea behind bootstrapping is that the bootstrap samples represent random permutations of the data that mimic the behavior of the CUSUM with no change-points. For each bootstrap sample k, we compute the change score S^k_{diff}. The confidence level is then computed by identifying the fraction of bootstrap samples that have change score less than S_{diff}. We use 99% confidence level to determine a significant change.

3. *Recursive detection:* In order to detect multiple change-points in an event-series, we recursively apply steps 1 and 2 to the two sub-series before and after the change-point until no more significant change-points are observed.

Fig. 2 shows an example of applying CUSUM to recursively detect multiple change-points in a time-series (level-shifts). The top, middle and bottom figures are the original time-series, ranked samples and the cumulative sums, respectively. The first and second change-points correspond to a positive level-shift and the third change-point corresponds to a negative level-shift. The transient one-time anomalies are not detected, as desired.

Identifying operationally relevant changes: We apply the above rank-based CUSUM test on each performance event-series and output a list of change-points. If an event-series remains steady over time, then it outputs no significant change-points. A significant change-point indicates that there is a behavior change in the event-series around the change-point. The diversity of event-series (*e.g.*, router CPU and memory utilization are either bursty with high deviation or stable with minimal deviation and router syslogs have small frequency counts) makes it challenging to interpret the changes. While a statistical change (before and after the change-point) of a small magnitude (*e.g.*, CPU increasing by 1%) may be mathematically significant, the operations team may not care. To capture operationally meaningful behavior changes, we incorporate post-filtering on the change-points using the relative mean difference between the mean before the change point and the mean after. The before time interval is chosen from the current change point to the nearest past change-point and similarly the after interval is chosen from the current to the nearest future change-point. The filtering threshold is chosen based on network domain knowledge. For example, for CPU and memory utilization, we can set the relative mean difference filtering threshold to be 0.6%, which captures the change-points such that the mean after the change is 1.6 times the mean before the change. We discuss the selection of the threshold in Section 4.4.

3.4 Trigger Change-point Association

When associating triggers with the changes in performance, network operators often focus on the performance event-series at the same location or within some proximity of the trigger. This is based on the experience or the domain knowledge related to the potential impact of the triggers. We use a spatial proximity model (similar to the one used in NICE [26]) to capture the causal impact scope of the triggers. The proximity model is specified in terms of the number of hops h between the location of trigger and the performance event-series. In this paper, we mainly focus on the same location for the trigger and performance event-series (*i.e.*, $h = 0$). In the future, we plan to explore the impact of triggers at locations one or two-hops away. For example, with $h = 1$, MERCURY will identify the impact of a trigger at one router on performance event-series at neighboring one-hop away routers.

Having identified the spatial proximity, the next step is to ensure that the timestamp of performance change-points identified by CUSUM are not too far from the given trigger. If all the change-points are far from a trigger, then there is no evidence that the trigger causes any behavior change. Thus, by analyzing the co-occurrence of performance change-points and the triggers, we can detect triggers that induce behavior changes and eliminate all triggers that do not co-occur with the change-points. We specify a (configurable) maximum timing lag to determine whether a change-point and a trigger co-occur.

Some triggers might co-occur with change-points by chance. To eliminate such false positives, we only consider those triggers that co-occur with at least two behavior changes with consistent signs (*i.e.*, both positive, or both negative) at two different locations (*e.g.*, routers). For example, given a trigger such as BGP policy change and CPU increases on one router and CPU decreases on another, then it is less likely that the BGP policy trigger is the root-cause for CPU change and we can eliminate the trigger. Such spatial association thus helps reduce the false alarms.

3.5 Commonality Detection

Given the list of trigger / change-point pairs, the next step is to identify if there is commonality across different behavior changes. For example, operating system upgrade trigger and the change-points in router CPU utilization might be observable only on a specific group of routers that have same OS version, model numbers and vendors. Extracting common attributes for different changes is very helpful for the network operations team to further drill-down into the changes and determine their root-causes.

There are multiple attributes that can be associated with a router: location, operating system (OS) version, role, model, vendor, type of line cards, number of BGP peers, uplink or downlink interfaces, customers. The problem of identifying the common attributes for trigger / change-point pair is that of a search in a multi-dimensional space of attributes. Given n attributes and each attribute can take up to maximum v values, then there are up to v^n possible combinations. Enumerating all possible combinations in a brute-force fashion does not scale when there are a large number of attributes.

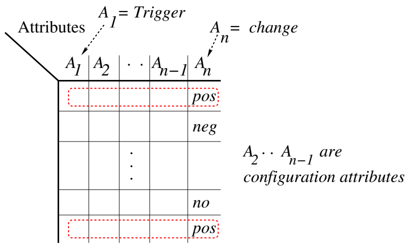

Figure 3: A matrix for specifying list of attributes.

Learning Commonality Rules. We propose to use a rule learner to automatically identify the common attributes. For each performance event-series, we first construct a matrix $[C]_{m\times n}$ where the n columns are the list of attributes and m rows provide the values for each router or device. We use the first column as the trigger and the last column to indicate the sign of the change (positive, negative or no change). Fig.3 illustrates the construction of the matrix. Incorporating the trigger in the matrix also helps in eliminating some of the triggers that do not have statistical evidence across multiple attributes to cause a behavior change.

The attributes can either be real-valued numbers or have nominal values (strings). Nominal values are useful in capturing attributes such as location, OS version, role, model, vendor and type of line cards. Other attributes like number of routing sessions, customers are real-valued. These real-valued attributes can take many values and identifying commonality then becomes even more challenging. We address this by first applying clustering on each real-valued attribute and identifying a small number of clusters. For example, some routers can have many BGP peers (*e.g.*, peering routers), whereas others have very few (*e.g.*, core routers). We use k-means clustering [13] to group the values and construct nominal values such as "less than x", "between x and y", and "greater than y".

We use RIPPER [6] as our rule learner for identifying the common attributes. RIPPER is a well-known fast machine learning algorithm that outputs a concise list of rules which are easier to understand and can deal well with noisy data. Some example rules learned by RIPPER are

1. if (trigger = OS upgrade) and (vendor = XYZ) then positive change in Memory
2. if (router_role = Border) and (trigger = BGP policy change) then positive change in CPU

The above rules provide an easy-to-interpret representation of the triggers and behavior changes across the network.

3.6 Aggregate Change Detection

For some KPIs, it is desirable to aggregate the event-series across multiple routers because they may be missed at individual routers (*e.g.*, due to too much background noise), and only manifest themselves at higher aggregation levels (such as network-wide aggregation or aggregated across certain types of routers). For example, given a BGP policy change trigger, the number of BGP timer expirations may not change too much for each router, but the total change might become significant across all the peering routers in the network. The goal of aggregation is to increase the significance of genuine changes, which are of interest to the network operations.

Spatial scope for aggregation: A key challenge is how to determine the subset of routers over which aggregation is performed. Instead of using arbitrary combinations of different attributes to define the aggregation level, we propose to use the rules learned by commonality detection (Section 3.5) to filter out individual routers that have already experienced either positive or negative changes. All the remaining routers experience no change according to these rules. We therefore aggregate event-series across all the remaining routers so that we can capture those changes that only manifest themselves at the higher aggregation level.

Time alignment of distributed triggers: The next step is to aggregate the event-series. We need to be careful in performing the aggregation because across different locations, the triggers can be applied at different times. For example, to avoid abrupt service disruption, an OS upgrade is typically applied to different routers in the network gradually over the course of several days (or even weeks). Such common practice can easily blur the precise location of the trigger and the corresponding change-points, resulting in missed detection (*i.e.*, false negatives). To account for the different times of distributed triggers across the network, we time align different instances of the same trigger for each KPI. Such time alignment ensures that after aggregation, any behavior change can be attributed to the same type of trigger. We then aggregate the time-aligned event-series by partitioning them into equal time-bins and computing the average within each time-bin.

Change point detection: We apply the same rank-based CUSUM test as described in Section 3.3 to detect changes in the aggregate event-series. If the change-point identified does not co-occur with the trigger (within a specified maximum timing lag), then we conclude that the trigger does not have an impact on the corresponding event-series at the higher aggregate level. On the other hand, if it does correlate, then the trigger might be the root-cause for the behavior change in the aggregated event-series.

4. MERCURY EVALUATION

In this section, we present evaluation of MERCURY based on data collected from a large tier-1 ISP network. First, we demonstrate that MERCURY is able to identify upgrade-related triggers of interest while eliminating the majority of configuration changes related to customer provisioning. We validate the results with the operators of the tier-1 ISP. Second, we show that MERCURY is able to detect behavior changes in network performance that are induced by the triggers. Finally, the commonality rules discovered by looking across a set of attributes associated with the routers is useful in better interpretation of the trigger and change point relationships. We further confirm some of the rules with network operations.

4.1 Data Set

As mentioned earlier, the tier-1 ISP collects a large set of data including router and line-card configurations, workflow logs, SNMP and router syslogs. We conducted our analysis using data collected over a six-month period. We focused on five categories of routers: core router (CR), aggregate router (AGG), access router (AR), route reflector (RR) and hub router (HR). A core router (CR) interconnects PoPs of the network. An aggregate router (AGG) aggregates traffic from multiple customers or routers and forwards them to the core routers. Access routers (AR) are the routers to which the ISP customers connect. A route reflector (RR) is used to peer with multiple BGP routers within an autonomous system (AS). VPNs typically have hub and spoke topology, where several spoke domains are connected to a hub router (HR).

We extract the configuration changes and operating system upgrades from the router configurations. We further group the configuration commands into sessions using the timestamp and the network operator's username as described in Section 3.2. Table 1 shows the performance data (collected via SNMP and syslog) from different categories of routers within the tier-1 ISP network. We construct the event-series for the router CPU and memory utilization metrics from SNMP and the normalized event-series from the router syslogs as described in Section 3.3.

4.2 Methodology

Evaluating MERCURY using real network data is challenging due to the lack of complete knowledge about the behavior changes

Router role	CR	AGG	AR	RR	HR
Perf. series	103,112	43,226	113,079	6,548	24,095

Table 1: Performance data collected over six months at a large tier-1 ISP network.

	Threshold	Output	False positives (FP)	FP rate
Config. sessions	2	2199	101	0.05
	4	6272	1095	0.17
	6	9562	2297	0.24
	8	12791	3873	0.30
	10	13581	4168	0.31
Triggers	2	120	4	0.03
	4	185	9	0.05
	6	212	17	0.08
	8	228	23	0.10
	10	236	27	0.11

Table 2: Number of configuration sessions and triggers output by MERCURY using the rareness metric and varying the rareness threshold from 2 to 10. The false positives are computed by comparing to customer provisioning sessions.

	Output	False positives (FP)	FP rate
Config. sessions	647	116	0.18
Triggers	92	24	0.26

Table 3: Number of configuration sessions and triggers output by MERCURY using the skewness metric. The false positives are computed by comparing to customer provisioning sessions.

induced by the upgrades or triggers in the network. We address this by interacting closely with the network operators, and confirming the results obtained by MERCURY.

We use the configuration records and workflow logs to identify the list of upgrades performed by the network operations. To test the effectiveness of our heuristics, we compare interesting triggers to the manually labeled upgrades by domain experts and to the ones that we know correspond to customer provisioning activities.

For quantifying the efficacy of detecting performance behavior changes induced by the interesting triggers, we share the rules learned by MERCURY with the network operations to confirm the earlier findings and detect any previously unknown behavior changes. We extract and normalize the performance event-series in SQL and Perl and implement our behavior change detection algorithms in Matlab. We use Perl to extract the interesting triggers from the workflow logs and router configuration snapshots. We use the JRip rule learner from the WEKA machine learning package [14] to discover the rules.

4.3 Validation of Triggers

In this section, we present our validation results for the interesting triggers identified by MERCURY from a very large set of network configuration changes. The goal is to demonstrate that MERCURY is able to detect network upgrades that the operations team consider important while filtering majority of unimportant configuration changes. After applying the rareness and skewness heuristics, MERCURY achieves an order of magnitude reduction in the number of sessions.

Comparison to labeled sessions related to customer provisioning (False positives). In the ISP network, there exist several automation tools for standard customer provisioning tasks (adding, removing, moving, upgrading customer specific router configurations). These tools share a common router login (or TACACS username) in the system. While not all customer provisioning activities are performed through these tools, almost all configuration changes made through these tools are customer provisioning related. Based on this knowledge, we evaluate the false positive rate of MERCURY by comparing its output against the configuration sessions carried through this special "username".

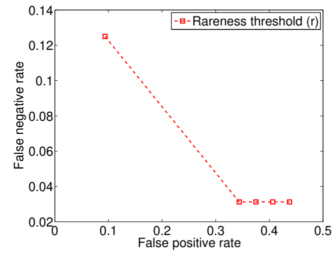

Figure 4: Evaluating sample network triggers output by MERCURY using domain expertise.

Table 2 shows the number of rare configuration sessions and triggers identified by MERCURY and the number of false positives for varying thresholds from 2 to 10 in increments of 2. For higher thresholds, we allow more configuration commands to be considered as rare and thus the number of sessions identified as rare is larger. The false positives are also higher for higher thresholds. Table 3 shows the number of skewed sessions and triggers identified by MERCURY and the number of false positives. We manually inspected the false positive triggers and observed that they make sense mathematically, however operations did not consider them as significant upgrades. We decided to keep the false positives in the subsequent step of behavior change detection.

Comparison to customer provisioning tools only allows us to quantify false positives. False negatives (important triggers missed by MERCURY) are a challenge to evaluate in large networks. We address this by asking the operators to manually label configuration sessions as what they think to be interesting, or non-interesting.

Comparison to triggers labeled by the network operations (False positives and false negatives). We provided a list of 32 triggers across different router locations, roles and vendors to the network operations team. The list captured different semantics in each type of trigger. 13 out of 32 are labeled as interesting and remaining 19 are considered to be non-interesting. By comparing the MERCURY output to the list of interesting and non-interesting triggers, we quantify the false positives as well as false negatives.

Fig. 4 shows the trade-off between false positives and false negatives as the threshold r for determining rareness of configuration sessions is varied from 2 to 10. For $r = 2$, false positive rate is lower but false negative rate is higher. This is because few commands are considered rare and hence there is a high chance that we will miss the interesting triggers. With increasing r, the false positives increase.

4.4 Validation of Behavior Changes in KPI induced by triggers

In this section, we validate the behavior changes detected in the key performance indicators (KPIs) caused by the network triggers. In the absence of ground truth, we compare our results to that of expectations from network operations.

Table 4 shows the sequence of steps applied in MERCURY change detection and commonality rule learning. We select the rareness threshold of 4 and the change-point operational significance threshold of 0.1 for CPU and memory utilization and 1 for syslogs. The change-point operational significance is determined using the relative mean difference around the change-point. We select these thresholds based on network domain knowledge and experience. For example, since the majority of event-series from syslogs have lower absolute values, we use a higher significance threshold of 1.

The number of triggers identified is 185 which we use to drive the behavior change detection on the performance event-series captured in SNMP (CPU, memory) and router syslogs. The number of event-series for CPU and memory utilization is 988 each and for syslogs is 288,084. The trigger KPI pairs to examine is very

Key Performance Indicator (KPI)	KPI Count	Trigger Count	Spatial Granularity	Trigger KPI Pairs	KPI Change Points (CP)	Operationally significant Change-points (CP)	Trigger, CP pairs after co-occurrence	Rules for triggers, CP and attributes	Unique cases triggers, attributes
CPU	988	185	Individual	182,780	3,262	1,546	338	10	10
MEMORY	988	185	Individual	182,780	10,704	475	160	4	4
SYSLOG	288,084	185	Individual	53,295,540	94,849	48,083	8,640	520	188

Table 4: Behavior change detection and rule learner results using MERCURY with rareness threshold of 4 and change-point operational significance threshold of 0.1 for CPU and memory and 1 for syslogs.

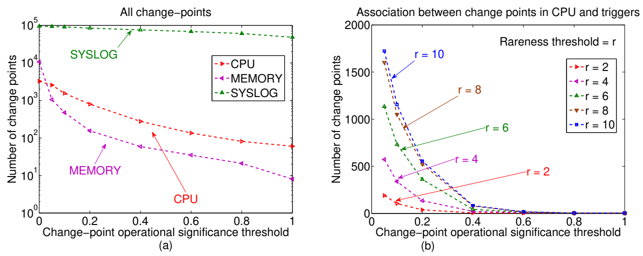

Figure 5: (a) Number of change-points in CPU, memory utilization and syslogs output by the recursive detector. (b) Number of change-points in CPU utilization that associate with the triggers detected by MERCURY (both rare and skewed sessions)

high implying that examining them manually is not feasible. For each KPI event-series, we identify change-points (CP) using the recursive detector and thus in some cases like CPU and memory, the number of change-points is greater than the number of event-series. Syslogs however, have less number of change-points. But due to the sheer volume of syslog messages, we still have around 94,849 changes-points in syslogs.

Many changes are genuine mathematically, but they are quite small to trigger any operations investigation. So, we use a ranking function on the relative mean differences for the change-points to prioritize them. This assists the operators to quickly focus on the top interesting behavior changes. For validating the changes in KPI induced by the triggers, we choose to ignore the small changes using the operational significance threshold. Fig.5(a) shows the number of change-points in router CPU, memory utilization and syslogs as the operational significance threshold is varied from 0.05 to 1. The threshold is determined using the relative mean difference between the before and after mean around the change-point. For higher change-point significance threshold, we filter out the changes with lower magnitude and capture only those with higher magnitude. We show the association between change-points in router CPU utilization and triggers identified by MERCURY in Fig.5(b). The curves for router memory utilization and syslogs are similar and not shown for space reasons. For higher rareness thresholds, since the number of rare sessions are higher, the number of change-points that associate with the triggers are higher.

We then use the association output between the trigger and change-point pairs as input to the RIPPER rule learner. We use 15 attributes associated with each router ranging from role, model, vendor, and location to number of routing protocols sessions and customers to detect commonality across behavior changes. The number of rules using triggers, change-points and attributes for CPU and memory are small (10 rules for CPU and 4 for memory). We inspected them and found all of them to be interesting behavior changes induced by the triggers. We further drilled down by visualizing some event-series on individual routers and indeed found significant behavior changes. The number of syslog rules, however, is still large (520) for data collected over a time interval of six months. We identified cases by grouping them into the number of unique triggers and attributes and found this to be quite manageable (188 cases). We also found some change-points for which we could not associate any trigger event. Such change-points might either be caused due to external network upgrades (in peer networks) or traffic changes. We consider the investigation of such change-points to future work.

	Trigger, KPI Pairs	KPI Change Points (CP)	Operationally Significant CP	Unique cases triggers
SYSLOG	53,295,540	5,846	3,181	92

Table 5: Aggregate change detection results for syslogs using time-alignment for distributed triggers and change-point operational significance threshold of 1.

Table 5 shows the aggregate changes in syslogs induced by the network upgrades (or triggers). The number of syslog event-series that have operationally significant changes is 3,181 which is a huge reduction from 53,295,540 trigger and KPI pairs. We identified the number of unique triggers to be 92 that induces aggregate change in at-least one event-series.

5. CASE STUDIES

In this section, we describe case studies on three types of network upgrades. In these case studies, MERCURY successfully revealed interesting network behavior changes caused by network upgrades. In all of these cases, we were analyzing historical events as a means of demonstrating the full power of MERCURY and how it would be applied in operational scenarios. We used MERCURY to validate expected performance improvements that were a direct result of network upgrades. We also applied MERCURY to detect performance changes which were not expected or designed outcomes of a given network upgrade. Given that this exercise used historical events, operations had in most situations already identified the impacts. However, in executing this evaluation, we also identified some smaller performance changes which had flown under operations' radar. Given this successful application of MERCURY on historical network upgrades, the operations team in the tier-1 ISP network is using MERCURY on an ongoing basis to monitor the impact of upgrades on network performance.

5.1 Impact of Router Operating System Upgrades

Our first case study focuses on Operating System (OS) upgrades on network routers. Router software is constantly evolving, with new features and bug fixes being incorporated on a regular basis. Installing new router software necessarily requires upgrading routers across the network -a tremendous task across potentially hundreds or even thousands of routers. Extensive lab testing is performed on each new software version before it is deployed, and then deployment is carefully and slowly ramped up in a bid to ensure that any potential latent issue is identified before scale deployment. New software versions often incorporate enhancements designed to improve network performance. For example, a router vendor may have incorporated enhancements in a new software version in a bid to reduce router CPU utilization, or to reduce the number of protocol flaps (e.g., BGP flaps). In such cases, it is imperative that the routers be carefully monitored after upgrades to validate that the expected improvements are indeed seen across the network. Similarly, as new software is deployed on network routers, it is imperative that these routers be carefully monitored for any potential performance degradations which may be indicative of new software bugs being introduced. Such conditions must be detected as rapidly as possible so as to minimize customer impact. However, manually monitoring the wide range of potential

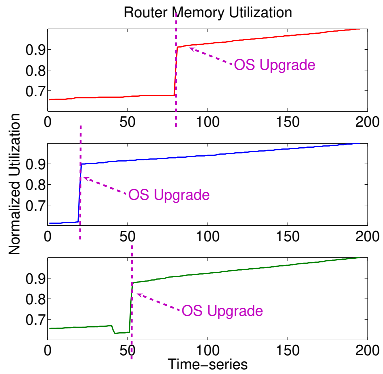

Figure 6: Case Study: Impact of router OS upgrades on memory utilization. Results shown for 3 aggregate routers (AGG). The utilization levels shown for each router are normalized by the maximum value in the six month interval.

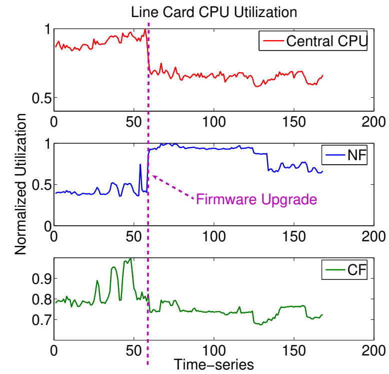

Figure 7: Case Study: Impact of router firmware upgrades on line card CPU utilization. Results shown for three line cards - central CPU, network-facing (NF), and CF (customer facing).

performance metrics across hundreds or thousands of routers over an extended period after the upgrade is a near impossible task. Despite significant efforts, it is unavoidable that issues fly under the radar - degrading customer performance yet eluding detection because of the immense scale of the issue at hand.

Given the complexity of router software, it can have impact on a wide range of different performance metrics indicative of network element health and customer performance. For example, poorly implemented software may result in higher than desired CPU or memory utilization. Both are critical and limited resources - if a router's CPU is heavily utilized by high priority processes, it may prevent routing protocol messages from being processed and thereby put the network control plane at risk. Similarly, if memory leaks are introduced in the software, the routers may be at risk of nasty crashes. Thus, it is absolutely critical that both resources be carefully managed. Router CPU and memory utilization are typically measured via SNMP at regular intervals (e.g., every 5 minutes). Metrics tracking the rate of link flaps, protocol session timeouts and other customer performance-impacting events also require careful monitoring to track improvements and detect degradations over time. The majority of these measures can be readily captured via router syslogs.

We apply MERCURY on CPU and memory utilizations in SNMP measurements and syslog messages on routers across the network and automatically identify behavior changes in CPU/memory utilization and in frequency of certain classes of syslog messages. We discuss here some of the significant and more interesting changes observed by MERCURY.

Findings:

1. Downticks in router CPU utilization. Using MERCURY, we observed significant improvements (downticks) in router CPU utilization across many router classes on recent upgrades. This improvement was a direct result of software enhancements implemented by the vendor to improve the router software efficiency. It was thus a designed and expected improvement, which MERCURY was able to validate at scale. Given the vast number of routers being upgraded, manual validation of router CPU utilization improvements would typically be achieved by sampling a subset of routers. However, MERCURY is able to analyze each and every router across the network. Thus, MERCURY can achieve far more thorough analysis than what is achievable by hand. The benefit was clearly evident as MERCURY identified some routers that contradicted the general trend, demonstrating significant degradations in CPU utilization. Specifically, some individual routers demonstrated a noticeable and somewhat concerning increase in CPU. Upon identification by MERCURY, operations was informed of these and vendor analysis was instigated.

2. Upticks in memory utilization on aggregate routers. For aggregate routers inside the backbone network, we discovered that there were upticks in router memory utilization after the OS upgrades. We show the time-series plots for three sample aggregate routers in Fig.6. The OS upgrades occur at different times across different routers, but each induces a behavior uptick in memory utilization. Network operations confirmed that this is a result of a larger OS image. The same increase in memory utilization was not observed across other router types with different models and OS versions and MERCURY automatically identified this.

3. Varying behaviors in layer-1 link flaps across different OS versions on access routers. In applying MERCURY to blindly analyze all distinct router syslog messages, we observed significant downticks in layer-1 link flaps as the routers were upgraded from OS version X to Y and then upticks when upgraded from Y to Z. On further investigation and interaction with the router vendors, the vendor confirmed that a bug had been introduced in version Y. Fortunately, this intermediate version (Y) had only been rolled out on very few routers.

4. Protection switching on access routers. Line card protection is used on access routers to protect customers from line card failures - if a line card fails, the customers are switched over to a dedicated backup line card. This switching is known as Automatic Protection Switching (APS). APS events are rare events - especially if looking at a single router as opposed to aggregated across the network. In analyzing a series of router upgrades, MERCURY successfully detected a behavior change (uptick) in the number of APS events across access routers. This observation confirmed that MERCURY could indeed detect critical issues that operations had also observed via manual tracking of events. Interestingly, MERCURY did not detect the changes on individual routers, but observed it in the aggregate time-series across routers. This was simply because the time-series was too sparse on a per router basis - meaningful statistical changes could only be observed at the aggregated level.

5.2 Impact of Firmware Upgrades

In our second case study, we apply MERCURY to detect the impact of firmware upgrades on the line card CPU utilization levels. The firmware upgrades are applied by the network operations on a per line card basis. Each router consists of multiple line cards - typically, two line cards in a router support the central CPU (used for tasks such as route computation) whilst other cards carry traffic. However, each of these line cards has a CPU whose utilization must be monitored. With 16 or more line cards per router, and up to thousands of routers, this is an extremely large number of line cards to monitor manually!

Findings: *Upticks on some optical carrier line cards.* We applied MERCURY on the line card CPU utilizations and observed that for

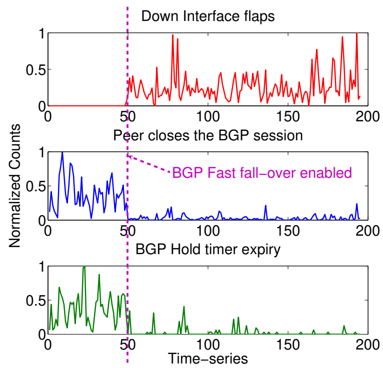

Figure 8: Case Study: Impact of BGP fast fall-over configuration change. Results shown for one access router (AR) and three event-types.

the majority of the line cards, there is a significant downtick in CPU utilization. However, on certain classes of routers (namely access routers), we observed a contradictory uptick in CPU utilization on specific types of line cards - specifically those which are network-facing. Fig. 7 shows the time-series plots for three line-card CPU utilization levels on the same router. As can be seen, there is a downtick on the central router CPU and the customer-facing cards, however an uptick on the network-facing cards draws attention for the network operator. Our current hypothesis is that some route computation processes might be migrated from the central router CPU to the network-facing line cards. Without MERCURY, the network operations team had been unaware of this behavior change. An investigation is now underway.

5.3 Impact of Configuration Changes

Our third case study focuses on understanding the impact of interesting configuration changes performed on routers within the tier-1 ISP network. As discussed in Section 4, we analyzed the ISP workflow (TACACS) logs to detect interesting configuration changes which correspond to network upgrades. We then systematically analyzed each of those triggers to identify those which caused noticeable behavioral changes. There were a wide range of interesting scenarios, a number of which are currently undergoing investigation. We show one example to demonstrate how MERCURY successfully detected the network upgrade and validated the intended performance impacts of these upgrades.

In this example, a configuration change in BGP fast external fall-over policy is categorized by MERCURY as a rare configuration session across a time interval of six months. This is because the fast external fall-over configuration is applied on a per-router basis - it is basically configured once per router to fundamentally change the behavior of the router on an ongoing basis. If fast external fall-over is enabled, the BGP peer immediately turns down the session when the layer-1 link goes down. On the other hand, if disabled, the peer waits for the default hold timer (3 keep-alives) to expire and then explicitly reset the peering session. Thus, fast external fall-over provides immediate response to layer one issues - achieving a faster failure recovery for these common events.

The act of turning on fast external fallover on a given router should result in a new set of BGP flaps that are specifically labeled as being induced by layer-one detection on the router. In Cisco syslogs, these are referred to "down interface flaps". The configuration change should also demonstrate a comparable degradation in the number of "timer expired" and "peer closed" events. Before the change, layer one induced flaps would have been detected via either "timer expired" or "peer closed" events, while after the configuration change, the L1 events would have been observed as "down interface flaps".

Findings: We observe the behavior change across all access routers that enabled the fast fall-over policy. Fig. 8 shows the time-series for BGP flaps, remote peer closing the BGP session and hold timer expiry on one router. The observation in Fig. 8 demonstrated that the fast external fallover configuration changes implemented in the network successfully created the desired behavioral change. As desired, the number of "down interface flaps" went from zero (when fast external fallover was not configured in the network) to a significant value. There was a corresponding decrease in the number of BGP hold timer and peer closed events. Thus, MERCURY proved that the routers were successfully detecting the layer one events and responding accordingly. This was consistent behavior across all configured routers - something which a manual inspection would not be able to scale to deduce. Furthermore, MERCURY could quantify the changes across all routers - as can be seen from Fig. 8, the majority of BGP flaps are induced by what the ISP router considers[1] to be layer-one induced failures.

6. RELATED WORK

We present related work on configuration analysis and performance troubleshooting and describe the differences with MERCURY.

Configuration Analysis. Recently, there has been a great deal of work [4, 10, 24, 27, 31] in understanding network configurations and modeling them to better understand and improve network designs. Misconfigurations are a common source of network problems. There are several proposals to detect and diagnose misconfigurations. PeerPressure [35] builds statistical model of healthy machines and compares to identify sick machines. Strider [36] uses state differencing to identify the root causes for different program behaviors. Chronus [37] uses virtual machine monitors, time travel testing and search to discover configuration errors. NetPrints[1] uses decision-tree learning for troubleshooting home network misconfigurations. Feamster *et al.* [11] use static analysis to discover configuration errors in BGP by testing constraints using high-level specification. WISE [32] provides a what-if analysis tool to estimate the effect of network configuration changes on service response times. Alimi *et al.* [2] propose shadow configurations to evaluate configuration changes. None of the above approaches describe how to automatically extract interesting upgrade-related configuration changes from the workflow logs and configurations. MERCURY does this using the rareness and skewness heuristics and focuses on trigger induced persistent behavior changes on a very large number of KPIs in real operational environments.

Performance troubleshooting. Detecting anomalies and troubleshooting network performance has a rich literature. PCA [15, 21, 22, 28] is one of the widely used technique for detecting network-wide anomalies. Zhang *et al.* [39] formalize different aspects of anomaly detection under a single framework. NetMedic [19] uses OS information to diagnose anomalies using the dependency structure between the components. Zhang *et al.* [40] use spatio-temporal compressive sensing to detect anomalies in traffic matrices. Xu *et al.* [38] use PCA to extract features from the console logs and decision trees to learn rules from the PCA output.

For troubleshooting, there have been several recent proposals using Bayesian network analysis and statistical mining techniques. SCORE [20] models the fault diagnosis problem using a bipartite graph and uses risk modeling to map high-level failure notifications into lower-layer root causes. Shrink [18] extends this model to deal with probabilistic settings. Sherlock [3] proposes a multi-level graph inference to discover the service-level dependencies in enterprise networks. Orion [5] uses delay timing analysis to discover traffic dependencies. eXpose [17] uses spectral graph parti-

[1]We say, "considers to be here" because certain failures on a remote router, such as hardware failures cannot be distinguished locally from layer one network failures.

tioning and mutual information to discover communication rules in edge networks. [8, 9, 16] uses network tomography to identify the location of the failures. Minerals [23] mines correlation patterns using association data mining. NICE [26] focuses on detecting and troubleshooting undesirable chronic network conditions using statistical correlations. Giza [25] applies multi-resolution techniques to localize regions in IPTV network with significant problems and l_1-norm minimization to discover causality between event-series. URCA [29] uses feedback from the anomaly detector to eliminate flows that exhibit normal behavior. ASTUTE [30] is a traffic anomaly detector that uses the equilibrium property and correlation across anomalous flows to discover a new class of anomalies.

MERCURY differs from all of the above described approaches in its application of analyzing the impact of upgrades on network performance. It focuses on detecting persistent behavior changes in network performance as opposed to short-term transient anomalies. The time-alignment for distributed triggers is a novel technique in MERCURY, not explored by any of the previous approaches.

7. CONCLUSIONS AND FUTURE WORK

We presented the design and implementation of MERCURY, a novel system for monitoring the performance impact of network upgrades (or triggers) in large operational networks. MERCURY uses the rareness and skewness properties of configuration to identify a small list of interesting triggers. It detects behavior changes in performance event-series using a rank-based statistical test. It uses a rule learner to identify commonality across the changes at multiple locations. It uses a novel time-alignment approach for distributed triggers to identify aggregate changes. We have applied MERCURY using real network data collected from a large tier-1 ISP network. Our results demonstrate that MERCURY is able to identify interesting triggers and behavior changes induced by the triggers. On multiple occasions, MERCURY also discovers previously unknown behavior changes, highlighting its ability to identify network conditions flying under the radar.

In the future, we plan to extend MERCURY in several directions. First, we would like to extend the capability of identifying triggers by closer collaborations with the network operations team. Second, we plan to expand MERCURY to detect behavior changes caused by chronic failures or external network conditions. Finally, we would like to extend MERCURY to discover behavior changes in correlation structures across different event-series. This is important when each individual performance event-series does not undergo significant changes, but the joint distribution does.

Acknowledgement

We thank the Sigcomm anonymous reviewers and our shepherd Pablo Rodriguez for their feedback. We are grateful to the network operations team for their invaluable help on the data collection, the case-study analysis and application of MERCURY. We also thank Mukta Mahimkar for suggesting the name MERCURY. This work is supported in part by NSF grants CNS-0546720, CNS-0615104, and CNS-0627020.

8. REFERENCES

[1] B. Aggarwal, R. Bhagwan, V. N. Padmanabhan, and G. Voelker. NetPrints: Diagnosing home network misconfigurations using shared knowledge. In *NSDI*, 2009.

[2] R. Alimi, Y. Wang, and Y. R. Yang. Shadow configuration as a network management primitive. In *Sigcomm*, 2008.

[3] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Sigcomm*, 2007.

[4] T. Benson, A. Akella, and D. Maltz. Unraveling the complexity of network management. In *NSDI*, 2009.

[5] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl. Automating network application dependency discovery: Experiences, limitations, and new solutions. In *OSDI*, 2008.

[6] W. W. Cohen. Fast effective rule induction. In *Machine Learning*, 1995.

[7] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Diamond in the rough: finding hierarchical heavy hitters in multi-dimensional data. In *Sigmod*, 2004.

[8] I. Cunha, R. Teixeira, N. Feamster, and C. Diot. Measurement methods for fast and accurate blackhole identification with binary tomography. In *IMC*, 2009.

[9] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot. Netdiagnoser: troubleshooting network unreachabilities using end-to-end probes and routing data. In *CoNEXT*, 2007.

[10] W. Enck, P. McDaniel, S. Sen, P. Sebos, S. Spoerel, A. Greenberg, S. Rao, and W. Aiello. Configuration management at massive scale: system design and experience. In *ATC*, 2007.

[11] N. Feamster and H. Balakrishnan. Detecting bgp configuration faults with static analysis. In *NSDI*, 2005.

[12] C. Finseth. An access control protocol, sometimes called tacacs, 1993. `http://www.faqs.org/rfcs/rfc1492.html`.

[13] E. Forgey. Cluster analysis of multivariate data: Efficiency vs. interpretability of classification. In *Biometrics*, 1965.

[14] S. R. Garner. WEKA: The waikato environment for knowledge analysis. In *New Zealand Computer Science Research*, 1995.

[15] Y. Huang, N. Feamster, A. Lakhina, and J. J. Xu. Diagnosing network disruptions with network-wide analysis. In *Sigmetrics*, 2007.

[16] Y. Huang, N. Feamster, and R. Teixeira. Practical issues with using network tomography for fault diagnosis. *SIGCOMM CCR*, 2008.

[17] S. Kandula, R. Chandra, and D. Katabi. What's going on? learning communication rules in edge networks. In *Sigcomm*, 2008.

[18] S. Kandula, D. Katabi, and J.-P. Vasseur. Shrink: A tool for failure diagnosis in IP networks. In *MineNet*, 2005.

[19] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed diagnosis in enterprise networks. In *Sigcomm*, 2009.

[20] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. IP fault localization via risk modeling. In *NSDI*, 2005.

[21] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *Sigcomm*, 2004.

[22] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *Sigcomm*, 2005.

[23] F. Le, S. Lee, T. Wong, H. S. Kim, and D. Newcomb. Minerals: using data mining to detect router misconfigurations. In *MineNet*, 2006.

[24] F. Le, G. G. Xie, D. Pei, J. Wang, and H. Zhang. Shedding light on the glue logic of the internet routing architecture. 2008.

[25] A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao. Towards automated performance diagnosis in a large IPTV network. In *Sigcomm*, 2009.

[26] A. Mahimkar, J. Yates, Y. Zhang, A. Shaikh, J. Wang, Z. Ge, and C. T. Ee. Troubleshooting chronic conditions in large IP networks. In *CoNEXT*, 2008.

[27] D. A. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjálmtýsson, and A. Greenberg. Routing design in operational networks: a look from the inside. In *Sigcomm*, 2004.

[28] H. Ringberg, A. Soule, J. Rexford, and C. Diot. Sensitivity of PCA for traffic anomaly detection. In *Sigmetrics*, 2007.

[29] F. Silveira and C. Diot. URCA: Pulling out anomalies by their root causes. In *INFOCOM*, 2010.

[30] F. Silveira, C. Diot, N. Taft, and R. Govindan. ASTUTE: Detecting a different class of traffic anomalies. In *SIGCOMM*, 2010.

[31] Y.-W. E. Sung, C. Lund, M. Lyn, S. G. Rao, and S. Sen. Modeling and understanding end-to-end class of service policies in operational networks. In *Sigcomm*, 2009.

[32] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar. Answering what-if deployment and configuration questions with WISE. In *Sigcomm*, 2008.

[33] W. A. Taylor. Change-point analysis: A powerful new tool for detecting changes. 2000. `http://www.variation.com/cpa/tech/changepoint.html`.

[34] R. S. Tsay. Outliers, level shifts and variance changes in time series. In *Journal of Forecasting*, 1988.

[35] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang. Automatic misconfiguration troubleshooting with PeerPressure. In *OSDI*, 2004.

[36] Y.-M. Wang, C. Verbowski, J. Dunagan, Y. Chen, H. J. Wang, C. Yuan, and Z. Zhang. Strider: A black-box, state-based approach to change and configuration management and support. In *LISA*, 2003.

[37] A. Whitaker, R. S. Cox, and S. D. Gribble. Configuration debugging as search: Finding the needle in the haystack. In *OSDI*, 2004.

[38] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting large-scale system problems by mining console logs. In *SOSP*, 2009.

[39] Y. Zhang, Z. Ge, A. Greenberg, and M. Roughan. Network anomography. In *IMC*, 2005.

[40] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu. Spatio-temporal compressive sensing and internet traffic matrices. In *Sigcomm*, 2009.

[41] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund. Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In *IMC*, 2004.