OnTimeDetect: Dynamic Network Anomaly Notification in perfSONAR Deployments

Prasad Calyam, Jialu Pu, Weiping Mandrawa, Ashok Krishnamurthy

Ohio Supercomputer Center/OARnet, The Ohio State University,
{pcalyam, jpu, ashok}@osc.edu; mandrawa@oar.net

***Abstract*—To monitor and diagnose bottlenecks on network paths used for large-scale data transfers, there is an increasing trend to deploy measurement frameworks such as perfSONAR. These deployments use web-services to expose vast data archives of current and historic measurements, which can be queried across end-to-end multi-domain network paths. Consequently, there has arisen a need to develop automated techniques and intuitive tools that help analyze these measurements for detecting and notifying prominent network anomalies such as plateaus in both real-time and offline manner. In this paper, we present a dynamically adaptive plateau-detection (APD) scheme and its implementation in our "OnTimeDetect" tool to enable consumers of perfSONAR measurements within the data-intensive scientific communities in overcoming their existing limitations of network anomaly detection and notification. We empirically evaluate our APD scheme in terms of accuracy, agility and scalability by using measurement traces collected by OnTimeDetect tool from worldwide perfSONAR deployments in HPC communities.**

I. Introduction

Scientific communities involved in major simulation and experimental activities involving the Teragrid [1] and Large Hadron Collider (LHC) [2] are generating massive data sets on a regular basis. These data sets are being transferred to various labs and universities at several Gbps speeds on networks that span across continents. Given the real-time consumption demands of the data and the substantial investments made for the network infrastructure to support the data transfers, there is a rapidly increasing trend to deploy instrumentation and measurement frameworks such as perfSONAR [3]. These frameworks assist in measurement data collection, storage and dissemination for monitoring and diagnosing bottlenecks that hinder end-to-end data transfer speeds. In addition, the frameworks help in provisioning end-to-end network measurements that can be used to understand how network performance changes due to end-user behavioral patterns (e.g., high volume flows, flash crowds), network fault events (e.g., misconfigurations, outages, malicious attacks) and cross-traffic congestion impact end-application and protocol behavior [4].

The emergence of perfSONAR deployments in the scientific communities that rely on high performance computing (HPC) resources is recent and revolutionary in the area of network troubleshooting. This is because the deployments support web-services to publish and subscribe *multi-domain* network measurements of various network health metrics such as bandwidth, delay, jitter and loss. The web-service schemas have been standardized in the Open Grid Forum [5] and have been adopted in frameworks such as OSCARS [6], Cricket-SNMP [7], and PingER [8].

Numerous perfSONAR deployments are sampling both active and passive measurements of various metrics several times a day. They are exposing these collected measurements via web-services in the form of vast data archives of current and historic measurements on national and international backbones (e.g., ESnet, Internet2, GEANT, SWITCH). The consumers of these measurements (e.g., network operators, researchers in scientific disciplines) are faced with the challenge to analyze and interpret the vast measurement data sets across end-to-end multi-domain network paths with minimal human inspection. They direly need automated techniques and intuitive tools to query, analyze, detect and notify prominent network performance anomalies such as plateaus that hinder data transfer speeds. Timely and accurate anomaly notification can lead to quicker resolution of network faults, and thus proactively prevents end-users experiencing annoyingly slow data transfers on well-provisioned high-speed networks. In addition, they need effective techniques and tools to support network anomaly detection and notification in both real-time and offline manner at multi-resolution timescales.

There have been several recent network anomaly detection studies that utilize various statistical and machine learning techniques such as: principal component analysis [9], wavelet analysis [10], Kalman filter [11], covariance matrix analysis [12], and online change-point detection [13] [14]. Of these schemes, a plateau-detector scheme [13] that belongs in the change-point detection category has been found to be relatively more effective [15] for routine network performance monitoring due to its low complexity, ease of implementation and effectiveness. It has been adopted by large-scale monitoring deployments such as NLANR AMP [13] and SLAC IEPM-BW [14], which can be considered as predecessors to the perfSONAR deployments.

However, faithful adoption of the plateau-detectors used in the predecessors has limitations for perfSONAR users. The NLANR AMP and SLAC IEPM-BW deployments used *static* configurations of the salient threshold parameters in the variants of the plateau-detection schemes they employed. Specifically, they used static configuration of "sensitivity" and "trigger elevation" threshold parameters for increasing the probability of anomaly event detection while at the same time, decreasing the probability of false alarms. The sensitivity parameter is used to specify the magnitude of plateau change

This material is based upon work supported by the Department of Energy under Award Number: DE-SC0001331. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

that may result when an anomaly event on a network path is to be triggered. The trigger elevation parameters are used to temporarily increase the thresholds to avoid repeated triggers for a brief period after an anomaly event has been detected. Due to the static configurations in earlier frameworks, the sensitivity and trigger elevation threshold parameters of the *static plateau-detection* (SPD) schemes need to be manually calibrated to *accurately* detect plateaus in different measurement sample profiles on network paths. Such a laborious process for accurate anomaly detection is impractical for large-scale monitoring infrastructures involving large numbers of end-to-end network paths with dynamically changing traffic characteristics. Another anomaly detection short-coming in earlier large-scale monitoring deployments is that the SPD scheme implementations were *embedded* within their deployment software. Hence, SPD implementations are not amenable for network paths monitoring customization and drill-down analysis of anomaly events by consumers of web-service enabled perfSONAR measurement data sets.

In this paper, we present *a novel, dynamically adaptive plateau-detection (APD) scheme and its implementation in our "OnTimeDetect" tool to enable the rapidly growing consumers of perfSONAR measurements (e.g., network operators, researchers in scientific disciplines) within the data-intensive scientific communities in overcoming the existing limitations of network anomaly detection and notification*. To dynamically configure thresholds for the "sensitivity" and "trigger elevation" parameters in our APD scheme, we apply *reinforcement learning* that guides the learning process based on partially available Markov Decision Processes [12]. Our premise is that the raw measurements just after an anomaly event provide direct intelligence about the anomaly event itself, and leveraging them for reinforcement of the machine learning (to compare statistics of historic and current measurement samples for detecting change points) can make the anomaly detection more robust and accurate. Based on a systematic study of anomaly events in real and synthetic measurement traffic traces, we observe that *variance* of the network performance just after an anomaly event is the critical statistic that can be used for reinforcement to accurately trigger an anomaly. Leveraging this observation, we derive closed-form expressions using statistical curve-fitting principles for dynamically determining thresholds of sensitivity and trigger elevation parameters in our APD scheme. Thus, our APD scheme avoids manual calibration of sensitivity and trigger elevation threshold parameters used in SPD schemes for different profiles of measurement samples on network paths. It achieves low false alarm rates at the cost of a fractional increase in detection time that is needed for the reinforcement learning.

We empirically evaluate our APD scheme using an implementation of the "OnTimeDetect" tool that we have developed as an added contribution of this paper. The OnTimeDetect tool capabilities include: (i) anomaly monitoring customization of large measurement topologies, and (ii) drill-down analysis of anomaly events in vast measurement archives in both real-time and offline manner at multi-resolution timescales. We remark that the OnTimeDetect tool is the first to perform a large-scale network performance anomaly detection analysis leveraging perfSONAR web-services. We use the OnTimeDetect tool to query vast measurement data archives of current and historic measurements from several worldwide perfSONAR deployments. More specifically, *the real-network measurement data in this paper has been queried by our OnTimeDetect tool from 65 sites that monitor approximately 480 network paths connecting various HPC communities (i.e., universities, labs, HPC centers) over high-speed network backbones that include ESnet, Internet2, GEANT, CENIC, KREONET, LHCOPN and many others*. Our evaluation metrics include anomaly detection *accuracy*, *agility* and *scalability*.

Through our accuracy evaluations, we show that our APD scheme outperforms the SPD scheme in detecting anomaly events and avoiding false alarms. We compare the APD and SPD scheme accuracy results in terms of the three metrics: *success ratio*, *false positive ratio*, and *false negative ratio*. Note that a false-positive trigger is one that gets reported when there is no actual anomaly event, whereas, a false-negative trigger is one that does not get reported when in fact there is an actual anomaly event.

Next, through our agility evaluations, we show how the *anomaly detection times* using our APD scheme varies across perfSONAR deployments, and is on the order of one or more days. The reason for these variations and long detection times is mainly due to the currently chosen average periodic (a.k.a. stratified random) sampling patterns and frequencies in the perfSONAR deployments. We demonstrate that by using adaptive sampling in perfSONAR deployments, we can reduce the anomaly detection times to be on the order of a few hours, and inturn lessen the detection time tradeoff that is needed for reinforcement learning in our APD scheme.

Lastly, through our scalability evaluations, we calculate the average *analysis time* per-site to sequentially detect anomalies on a large number of perfSONAR monitored paths. We observe that the analysis times are on the order of tens of seconds for multi-resolution timescale queries. We define analysis time for a measurement archive as the sum of the times taken for perfSONAR web-service processing, measurement archive transfer and run time of the APD scheme on the measurement archive. We demonstrate that by using a parallel-query mechanism in the OnTimeDetect tool, we can speedup the average analysis time per-site by (40 - 60)%, and inturn improve the OnTimeDetect tool user experience during online drill-down analysis.

The remainder paper organization is as follows: Section 2 describes related work. Section 3 explains the SPD scheme and illustrates its limitations. Section 4 details our APD scheme. Section 5 discusses our OnTimeDetect tool implementation of the APD scheme and the performance evaluation with perfSONAR datasets. Section 6 concludes the paper.

II. Related Work

It is common practice even today to rely on user-defined monitor thresholds to detect and notify anomalies [7]. The thresholds can be exact values (e.g., notify anomaly if delay exceeds 50ms on a network path) or relations that use differences between temporally distinct values for the same data source or a secondary data source (e.g., notify anomaly

if difference in TCP throughput changes by 25%). Obviously, such user-defined threshold based methods do not consider any network path's inherent behavior, which has unique variability at any given time instant. Hence, they are prone to high false alarm rates. To adapt the anomaly detection to handle the unique variability on a network path, mean $\pm$ standard deviation (MSD) methods have been employed, where the thresholds are determined based on a moving window summary of raw measurements. However, such anomaly detection methods are not robust to outliers that are common in network measurements. The outliers are typically caused by intermittent spikes, intermittent dips, and bursts in network performance, which are typically not of interest as anomaly events.

The authors in [9] use principal component analysis principles and the authors in [10] uses wavelet analysis for detecting network performance anomalies. Both these studies involve offline analysis and focus on detecting anomalies in passive measurements at a network link basis. Hence, they are not suitable for real-time monitoring of anomalies pertaining to active measurements across end-to-end network paths, which are of greater interest in the context of bulk-data transfer application flows that traverse multiple links. The authors in [11] attempt to overcome the link basis limitation by creating a traffic matrix of all links within an enterprise network domain. They employ a Kalman-filter based anomaly detection scheme that filters out the network norm by comparing future predictions of the traffic matrix state to an inference of the actual traffic matrix obtained from recent measurements. The limitation of employing such a scheme in data-intensive communities arises due to the fact that bulk-data transfer application flows traverse multiple links across multiple domains. Hence, the traffic matrix sizes can become unwieldy very quickly. Moreover, it is not generally feasible to obtain link-level information in multi-domain scenarios due to policy limitations.

In [17], an adaptive fault detector algorithm is proposed that determines the baseline for network norm in local area networks using a stochastic approximation of the maximum likelihood function of recent performance samples. Abrupt jumps in means of local area network metrics (e.g., ARP broadcast packets, TCP connections to a web-server) are treated as anomalies. This scheme that is based on online change-point detection is not suitable for wide area network monitoring because it assumes that the data sets comprise of *k*-variate Gaussian distributions that are common in local area network data sets. Authors in [18] also use sequential change point detection principles to detect anomalies when correlated changes occur in various network health metrics collected along a network path.

There are also several related studies that use machine learning techniques for unsupervised anomaly detection [19] [20] [13] [14]. Frameworks such as NLANR AMP [13] and SLAC IEPM-BW [14] have used variants of the plateau-detector because it is well suited for real-time monitoring of anomalies pertaining to active measurements across end-to-end network paths. Specifically, it uses unsupervised machine learning that relies on raw measurements and dynamic network norm estimation. It also has low complexity, and is relatively easy to implement in large-scale network monitoring frameworks. Moreover, it has been shown to be reasonably effective for monitoring anomalies of active measurements comprising of end-to-end paths across multi-domain paths in the NLANR AMP and SLAC IEPM-BW frameworks for measurements collected over several years.

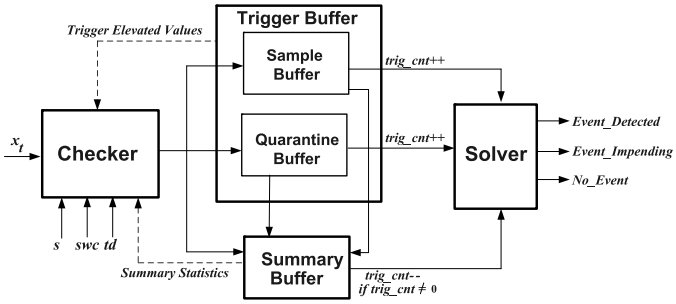


Fig. 1. Plateau-detector block diagram

III. Plateau Anomaly Detection

A. Plateau-Detector Overview

The description of the basic plateau-detector algorithm implemented in [13] and [14] is as follows. The algorithm is an enhanced version of the MSD method. In the MSD method, the first step is to determine the network health norm by calculating the mean μ for a set of measurements sampled recently into a "summary buffer". The sampled measurements correspond to network health metrics such as throughput, delay, loss on a network path. The number of samples in the summary buffer is user-defined and is specified using a "summary window count" swc. In the next step, an anomaly is triggered if the value of the most recent measurement sample x_t crosses either of the $\mu \pm \sigma$ thresholds of the summary buffer measurements; note that σ corresponds to standard deviation of the measurements in the summary buffer. In comparison to the MSD method, the plateau-detector requires two additional user-defined inputs called "trigger duration" td and "sensitivity" s. The trigger duration td specifies the duration of the anomaly event before a trigger is signaled. The sensitivity s specifies the magnitude of the plateau change that may result when an anomaly event on a network path is to be triggered.

Figure 1 shows the different components of a plateau-detector. The values of x_t are first input to a "checker" which compares whether the most recent x_t value lies within the upper and lower threshold set $ts(.) = \{T_{SU}, T_{QU}, T_{SL}, T_{QL}\}$ of the: (i) "summary buffer" *sumbuff* i.e., T_{SU} and T_{SL} or (ii) "quarantine buffer" *qbuff* i.e., T_{QU} and T_{QL}. These thresholds are illustrated in Figure 3 and are calculated using mean μ and standard deviation σ of the summary window as shown in Equations (1) - (4).

$$T_{SU} = \mu + s * \sigma \quad (1)$$

$$T_{QU} = \mu + 2 * s * \sigma \quad (2)$$

$$T_{SL} = \mu - s * \sigma \quad (3)$$

$$T_{QL} = \mu - 2 * s * \sigma \quad (4)$$

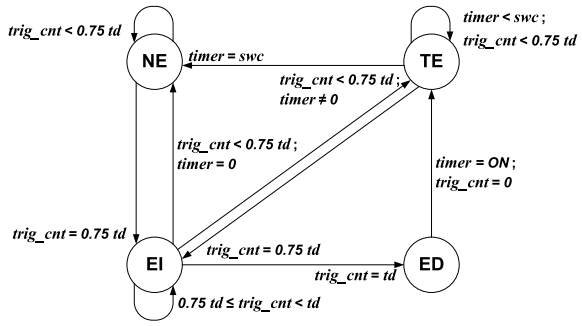


Fig. 2. Plateau detector states and state-transitions

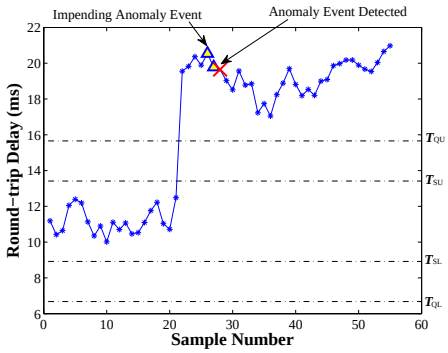


Fig. 3. Plateau-detector thresholds illustration

If x_t values lie within these thresholds, the plateau-detector will be in the no event (NE) state shown in Figure 2. In this state, x_t values are put into the *sumbuff*. If x_t values go below T_{QL} or exceed T_{QU}, they are put into the quarantine buffer *qbuff*. Similarly, if x_t values cross T_{SL} and T_{SU}, they are put into the "sample buffer" *sampbuff*. If x_t is put into either *qbuff* or *sampbuff*, a trigger count $trig\_cnt$ counter is incremented. Whereas, if x_t is put into *sumbuff*, $trig\_cnt$ is decremented as long as $trig\_cnt$ is non-zero. If $trig\_cnt$ exceeds 0.75*td due to increasing number of x_t values going into *qbuff* or *sampbuff*, then the plateau-detector enters into an event impending (EI) state. If the $trig\_cnt$ drops below 0.75*td, then the plateau-detector returns to NE state. Otherwise, the plateau-detector stays in the EI state until $trig\_cnt$ equals td, after which it enters into an event detected (ED) state. Figure 3 shows an anomaly event being triggered after x_t crosses the $ts(.)$ thresholds. The EI state x_t values are marked as triangles, and the triggered event in the ED state is marked by the cross mark. At this point, the $trig\_cnt$ is reset, and a *timer* is turned ON. The plateau-detector now goes into a trigger elevated (TE) state, where the values in the upper and lower threshold set $ts'(.) = \{T'_{SU}, T'_{QU}, T'_{SL}, T'_{QL}\}$ are calculated as shown in Equations (5) - (8).

$$T'_{SU} = 1.2 * max(x_t) \text{ in } \textit{trigbuff} \quad (5)$$
$$T'_{QU} = 1.4 * max(x_t) \text{ in } \textit{trigbuff} \quad (6)$$
$$T'_{SL} = 0.8 * min(x_t) \text{ in } \textit{trigbuff} \quad (7)$$
$$T'_{QL} = 0.6 * min(x_t) \text{ in } \textit{trigbuff} \quad (8)$$

Until the *timer* equals swc, the elevated thresholds are used for comparing x_t. The reason for the trigger elevation is to avoid reporting of repeated triggers for the already detected anomaly. It is relevant to note that the plateau-detector can transition from TE state to EI state if another anomaly occurs due to x_t crossing the elevated thresholds. Once *timer* equals swc, and x_t does not cross the elevated thresholds, the plateau-detector returns to the NE state. Referring back to the Figure 1, we can see that the "solver" tracks the plateau-detector state transitions and outputs the *No_Event*, *Event_Impending*, and *Event_Detected* signals.

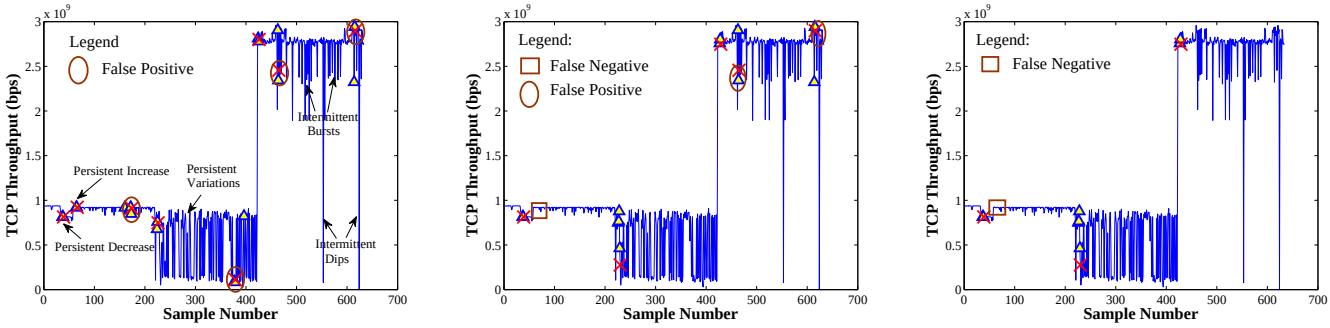
B. Plateau Detector Parameters

We now discuss selection of values used for the following plateau-detector parameters: summary window count swc, trigger duration td, sensitivity s, NE state threshold set $ts(.)$, and TE state threshold set $ts'(.)$. Based on this discussion, we motivate the need for dynamically adaptive determination of s, $ts(.)$, and $ts'(.)$.

The value of swc is chosen depending upon the number of recent history samples that are sufficient to obtain a rough yet reliable estimate of the network norm. Choosing a small value for swc has the risk of allowing network noise such as intermittent spikes, intermittent dips, or bursts that distort the network norm estimation. Alternately, choosing a large value for swc has the risk of smoothening out trends of impending anomalies that inturn increases detection time or leads to false negatives. Both the earlier plateau-detector implementations [13] and [14] have used a setting of swc = 20. This value assumes average inter-sampling periods to be in the range of 3 to 8 hrs on any given network path, which is typically equivalent to the network status in the most recent 1 to 3 days in history, respectively.

The value of td is chosen to be relatively smaller than the swc. The smaller the value of td compared to swc, the faster a trigger will be signaled in the event of an anomaly. However, the td must be chosen to be large-enough such that intermittent spikes, intermittent dips, or bursts i.e., noise events in network health do not influence the trigger signaling and cause false alarms. Given the fact that samples that don't cross the $ts(.)$ thresholds reduce the $trig\_cnt$, values on the order of td 5 to 10 are suitable. In the earlier plateau-detector implementations, the td is assumed to be approximately 1/3rd swc (i.e., td = 7). Based on our systematic study of anomaly events in real and synthetic measurement traffic traces (we provide detailed descriptions of the trace sources in Section IV), we have found that the assumptions of swc = 20 and td = 7 are reasonable, and minor modifications to these settings have negligible influence in triggered false alarms.

However, we found that the s, $ts(.)$, and $ts'(.)$ parameters are relatively more salient, and minor modifications in their values selection significantly influences the anomaly detection accuracy. Chosing the s value needs consideration of trade-offs i.e., a small s value results in triggers for slight variations in network performance magnitudes, whereas a large s value could overlook actual anomalies that should be detected. Both the NLANR AMP [13] and SLAC IEPM-BW [14] frameworks choose s = 2, and analysis with a large number of real



(a) s = 2 (b) s = 3 (c) s = 4

Fig. 4. Impact of choosing static s values on anomaly detection accuracy

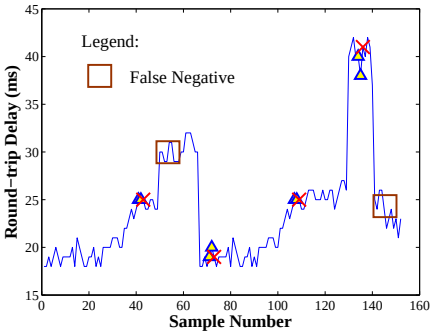


Fig. 5. Impact of choosing static $ts'(.)$ settings on anomaly detection accuracy

measurement traces in [15] has shown that s settings in the range of 2 to 3 is effective. In fact, even in grid environment monitoring of resources such as CPU, a variant of the MSD method with thresholds of $\mu \pm 2 * \sigma$, has been found to be effective [20]. In essence, setting of the threshold values to $\mu \pm 2 * \sigma$ is based on an inherent assumption that network norm data follows normal distribution. It is known that if any data follows normal distribution, approximately 95% of the data points fall within this $\mu \pm 2 * \sigma$ range. Since this assumption is not completely true, static setting of s = 2, which also directly impacts $ts(.)$ settings (see Equations (1) - (4)) causes false alarms. In our analysis of anomaly events in real and synthetic measurement traffic traces, we observed in some traces that minor changes in the s setting (e.g., s set to either 1.9, 2, 2.1, or 2.3) can result in a notable difference in false alarms.

Figures 4(a) - (c) show the impact of choosing static s values of 2, 3 or 4 for detecting anomaly events on a conjoined set of real-network traces. We conjoined the traces to illustrate how the anomaly detection accuracy is affected by the static s values for different kinds of network events such as persistent increase/decrease, persistent variations, intermittent spikes, intermittent dips, and bursts. As we can see, increasing the sensitivity from 2 to 3 reduces the number of false positives, and causes a false negative. Increasing the sensitivity to 4 can avoid the triggering of all false positives but the false negative remains. These observations motivate us to develop an adaptive scheme (described in detail in Section IV) for dynamically configuring s values, which avoids triggering such false alarms that occur when using the SPD scheme.

Similarly, we found that choosing static $ts'(.)$ settings as shown in Equations (5) - (8) can result in false alarms. Recall from Section III-A that the $ts'(.)$ settings use elevated thresholds based on the $max(x_t)$ and $min(x_t)$ values in the *trigbuff* until the $timer$ equals swc. Figure 5 shows how the static $ts'(.)$ settings do not detect consecutive anomalies of the same magnitude of either the persistent increase/decrease kinds when the $timer$ is lesser than swc. These observations motivate us to develop an adaptive scheme (described in detail in Section IV) for dynamically configuring $ts'(.)$ values, which avoids triggering false negatives in consecutive anomaly event scenarios when using the SPD scheme.

IV. Dynamically Adaptive Plateau-Detector Scheme

In this section, we describe the trace sources and our systematic study of anomaly events in real and synthetic measurements of network performance. In addition, we present our novel, dynamically adaptive plateau-detection (APD) scheme based on reinforcement learning to overcome the SPD scheme limitations described in Section III-B.

A. Trace Data

The real-network measurement trace sources for studying anomaly events referred in this paper are from several worldwide perfSONAR deployments [3]. Using customized perfSONAR web-service client scripts, we queried current and historic measurement data archives from 65 sites that monitor approximately 480 network paths connecting various HPC communities (i.e., universities, labs, HPC centers) over high-speed network backbones that include ESnet, Internet2, GEANT, CENIC, KREONET, LHCOPN and many others. Although we had the option to query a number of active and passive metrics, we queried only the active measurements of TCP throughput pertaining to the BWCTL tool [3] used in perfSONAR deployments. The BWCTL tool is a scheduling and policy daemon that wraps popular data throughput testing tools such as Iperf [21], Thrulay [22] and Nuttcp [23]. We

chose only to query BWCTL tool measurements of TCP throughput as an exemplar metric in this paper because: (a) it is commonly-used by consumers of perfSONAR meaurements, and (b) it has direct relevance to monitoring end-to-end network path performance affecting large-scale file transfers. Other measurements such as SNMP [7] only provide hop-status that is useful to know general network equipment health status. However, we remark that our observations and conclusions presented in this paper can be generally applied to any time-series of active or passive measurements.

To obtain greater flexibility in studying how plateau-detector parameters are affected by a wide-range of characteristics of anomaly events, we also use synthetic measurement traces. For generating the synthetic traces that mimic the behavior of real active measurement traffic traces, we leverage results from our earlier study on modeling real-network active measurement traces [24]. In this study, we analyzed large data volumes of active and passive measurements collected over campus, regional and national network backbones with time series trends that are: (i) routine, and (ii) event-laden [25], [26]. Our modeling was based on the Box-Jenkins method for time-series analysis [27]. Using diagnostics, 95% confidence interval checking, and prediction, we showed that it is reasonable to characterize active measurements on wide area network paths using Auto-Regressive Integrated Moving Average (ARIMA) model class parameters. We also showed that first-order differencing is sufficient to remove inherent trends and thus filter the data sets. Further, we concluded that active measurement time series have "too much memory", because of which they generally follow MA(q) process (even when they contain events with AR characteristics) with low q values. Hence, we generated synthetic traces for the round-trip delay metric following the *ARIMA* (0, 1, *q*) model with q values in the range of 1 to 3. Next, we injected the traces with a wide-range of anomaly events that had different white noise standard deviations in the anomaly time-series regions.

One of the significant challenges in dealing with all the measurement traces is to decide what kind of network events need to be labeled as anomaly events that have to be triggered by a detection scheme under test. For this task, we leveraged the established fact that plateau anomalies are of most interest since they have in most cases indicated reasons for changes in data transfer speeds on high-speed network paths [14]. In addition, network paths with high performance variability behave similar to noticeable plateau anomalies in terms of their statistical nature. Intermittent spikes, intermittent dips, and bursts generally are caused due to user-behavior during normal network operation, i.e., users generating various application traffic. Thus, intermittent spikes, intermittent dips, and bursts are not of interest as anomaly events and should not be notified. The anomalies we categorized as worthy of notification are based on our own experiences as network operators [25], [26], and are based on our extensive discussions with other network operators supporting HPC communities (e.g., ESnet, Internet2, GEANT). We remark that the causes for the performance patterns seen in the real-network measurement traces can be due to several reasons. It could be due to large-application flows, misconfigurations of network elements, cross-traffic congestion, or even due to changes in the test platforms (e.g., kernel driver, OS image, TCP flavor, auto-tuned buffers versus fixed buffers) of the active measurement probes.

B. Dynamic Sensitivity Selection

In our analysis of the above described real-network and synthetic traces, we found that s in the range of 2 to 4 consistently detected the labeled anomalies. We noted that although the SPD scheme frequently detected the labeled anomaly events, the false alarms i.e., false positivies and false negatives were caused primarily due to the s values not changing for various anomaly events. If the sensitivity at the time point of a false alarm was modified, the anomaly event was successfully detected. Hence, we concluded that s needs to be re-evaluated at each time step to avoid false alarms. Further, we observed that false alarms were triggered in the SPD scheme due to the nature of persistent variations in the time series after an anomaly event is detected. Hence, we concluded that variance σ_f^2 of the raw measurements just after an anomaly event provides direct intelligence about the anomaly event itself, and leveraging it for reinforcement of the machine learning (to compare statistics of historic and current measurement samples for detecting change points) can make the anomaly detection more robust and accurate. We remark that this idea of using σ_f^2 as the critical statistic for reinforcement to detect anomaly events robustly and accurately is formally referred to in the machine learning literature as "reinforcement learning". Also, finite-state Markov decision processes are commonly used to formulate the reinforcement learning.

Based on these observations, we use the $\frac{\sigma_f^2}{\sigma_c^2}$ relation in our APD scheme to determine the sensitivity s_{t-swc} dynamically at a time step t. Here, σ_f^2 refers to the variance of the swc number of measurement samples in the future, and σ_c^2 refers to the variance of the swc number of most current measurement samples. We set s_{t-swc} to a higher value (closer to 4) based on how much the $\frac{\sigma_f^2}{\sigma_c^2}$ ratio is higher than 1. Similarly, we set s_{t-swc} to a lower value (closer to 2) based on how much the $\frac{\sigma_f^2}{\sigma_c^2}$ ratio is lower than 1. Equations 9 - 12 show the expressions to calculate the σ_f and σ_c values.

$$\mu_c = \frac{1}{swc}\sum_{i=1}^{swc} swd[i] \tag{9}$$

$$\sigma_c = \sqrt{\frac{1}{swc-1}\sum_{i=1}^{swc}(swd[i]-\mu_c)^2} \tag{10}$$

$$\mu_f = \frac{1}{swc}\sum_{i=1}^{swc} rbd[i] \tag{11}$$

$$\sigma_f = \sqrt{\frac{1}{swc-1}\sum_{i=1}^{swc}(rbd[i]-\mu_f)^2} \tag{12}$$

Note that swd refers to summary window data and rbd refers to reinforcement buffer data. Figure 6 shows the

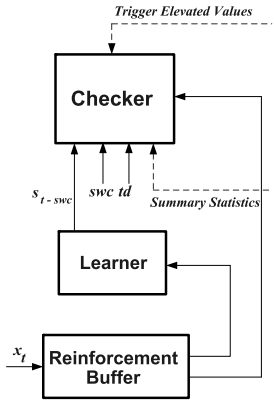


Fig. 6. Additional blocks needed in the plateau-detector for APD scheme

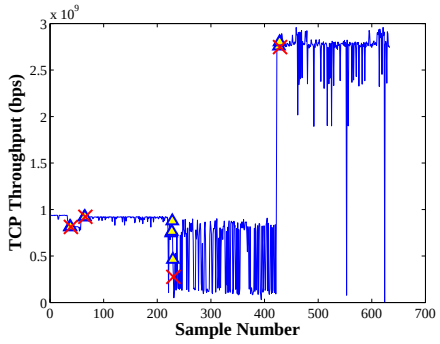


Fig. 7. Illustration to show choosing dynamic s values avoids false alarms

"Learner" and "Reinforcement Buffer" blocks needed in the plateau-detector shown in Figure 1 for dynamically determining the sensitivity parameter. The Learner implements our APD scheme logic to output instantaneous values of sensitivity s_{t-swc} to the Checker. The Reinforcement Buffer stores the measurement samples x_t as they arrive to the plateau-detector, which are subsequently used by the Learner for the reinforcement learning.

Using this setup of the plateau-detector, we record the $\frac{\sigma_f^2}{\sigma_c^2}$ values between sensitivity 2 to 4 that detect the different kinds of anomaly events accurately in various synthetic and real-network traces. Next, using statistical curve-fitting principles, we best fit the recorded values to derive the closed form linear expression as shown in Equation 13 for calculating the dynamic sensitivity s_{t-swc}.

$$s_{(t-swc)} = 0.4 * \frac{\sigma_f^2}{\sigma_c^2} + 2 \tag{13}$$

Figure 7 shows how the APD scheme accurately detects the anomaly events and avoids false alarms for the example trace described in Section III-B. Correspondingly, Figure 8 shows how the sensitivity configuration changes dynamically between the range of 2 to 4 during the robust and accurate anomaly detection of the APD scheme.

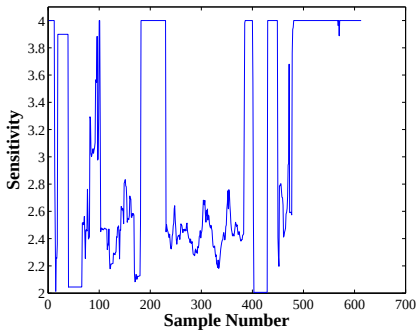


Fig. 8. Instantaneous s value selections in APD scheme

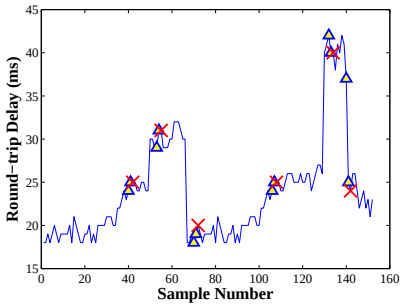


Fig. 9. Illustration to show choosing dynamic $ts'(.)$ settings avoids false alarms

C. Dynamic Trigger Elevation Selection

In our analysis of the synthetic traces, we found that choosing static $ts'(.)$ settings does not detect consecutive anomalies of the same magnitude of either the persistent increase/decrease kinds when the $timer$ is lesser than swc. We noted that using the $ts'(.)$ settings based on the $max(x_t)$ and $min(x_t)$ values in the SPD scheme was the cause for the false negatives. In order to understand how we overcame this limitation, let us consider the measurement sample x_d that arrived at the time when the anomaly was detected i.e., x_d refers to the x_t point on which we annotate the cross mark on the related graphs in this paper. By using x_d as the network norm in the trigger elevated (TE) state for calculating the thresholds, we were able to avoid the false negatives in consecutive anomaly event scenarios. Hence, we use Equations (14) - (17) to calculate the $ts'(.)$ upper and lower threshold set values.

$$T'_{SU} = x_d + s_t * \sigma_c \tag{14}$$
$$T'_{QU} = x_d + 2 * s_t * \sigma_c \tag{15}$$
$$T'_{SL} = x_d - s_t * \sigma_c \tag{16}$$
$$T'_{QL} = x_d - 2 * s_t * \sigma_c \tag{17}$$

Figure 9 shows how the false negatives that were marked in the Figure 5 for the SPD scheme are detected using the dynamic $ts'(.)$ settings in the APD scheme.

V. Performance Evaluation

In this section, we first describe our OnTimeDetect tool implementation of the APD scheme. Following this, we describe its use in the accuracy, agility and scalability evaluations of our APD scheme.

A. OnTimeDetect Tool Implementation

We have developed an OnTimeDetect tool prototype whose capabilities include: (i) anomaly monitoring customization of large measurement topologies, and (ii) drill-down analysis of anomaly events in vast measurement archives in both real-time and offline manner at multi-resolution timescales. We have developed both command-line and GUI versions of this tool that we plan to widely distribute in the near future with Windows and Linux platforms support. In the offline mode, a measurement trace with timeseries and measurement tuple can be loaded into the tool to obtain an annotated graph with anomalies. Various timeseries portions of the graphs can be zoomed in-and-out to analyze trends at different time granularities. The swc, td, s, $ts(.)$, and $ts'(.)$ parameters can also be adjusted to observe anomaly detection results. For any user setting of the parameters, the tool reports summary statistics of the measurement data being analyzed as well as the number of impending and anomaly events. The tool also allows a user to save the analyzed graphs. The graphs and anomaly report are intended to help in communicating network anomaly information in traces amongst end-users and network operators. In the online mode, the tool takes as input a site-list with each target site specified by its perfSONAR measurement archive webservice address e.g., http://wtg248.otctest.psu.edu:8085. This list can also be auto-generated for specific projects e.g., LHC and using the perf-SONAR global lookup web-service [3]. Upon specifying a query time resolution (e.g., 1 month), latest measurement data sets from the perfSONAR deployments are correspondingly queried and analyzed for anomalies using our APD scheme or for any static settings of the swc, td, s, $ts(.)$, and $ts'(.)$ parameters.

B. Accuracy Evaluation

We have extensively investigated the accuracy of our APD scheme on measurement traces collected by our OnTimeDetect tool from 65 sites that monitor approximately 480 network paths connecting various HPC communities. To illustrate the findings of our investigation, we choose to show the accuracy performance for a representative set of 8 traces shown in Table I with unique time series characteristics. We use three metrics shown in Equations 18, 19 and 20 respectively, to evaluate the anomaly detection accuracy of the APD scheme in comparison with the SPD scheme: *success ratio* (R_s), *false positive ratio* (R_{f+}), and *false negative ratio*(R_{f-}).

$$R_s = \frac{\textit{number of true triggers detected}}{\textit{number of true triggers}} \quad (18)$$

$$R_f+ = \frac{\textit{number of false triggers detected}}{\textit{number of true triggers}} \quad (19)$$

$$R_f- = \frac{\textit{number of true triggers missed}}{\textit{number of true triggers}} \quad (20)$$

Note that higher values of R_s and lower values of R_{f+} and R_{f-} denote superior performance. In addition, the value of the expression R_s+R_{f-} is always equal to 1. Table II shows the accuracy evaluation results for the APD scheme in comparison with the SPD scheme with static s settings of 2, 3 and 4. We can observe that the APD scheme outperforms atleast one of the static SPD schemes in detecting anomaly events and avoiding false alarms in all of the 8 traces. However, in some cases the APD scheme performs similar to a particular SPD scheme. The cases where the APD scheme outperforms the SPD scheme variants are shown in bold font. We can note that the SPD scheme with s=2 causes most false negatives indicating that it is least robust than the other schemes.

C. Agility Evaluation

Agility evaluations presented herein are for the purpose of showing the anomaly detection time results from our real-network measurement trace analysis. Our definition of detection time of an anomaly refers to the time difference between the instant the plateau-detector enters the impending (EI) state and the instant it is in the event detected (ED) state. As expected, low detection times indicates a superior anomaly detection scheme. Figure 10 shows the detection times for the representative set of 8 traces shown in Table I. We can observe that the anomaly detection times using our APD scheme varies across perfSONAR deployments, and is on the order of one or more days. The reason for these variations and long detection times is mainly due to the currently chosen average periodic (a.k.a. stratified random) sampling patterns and frequencies in the perfSONAR deployments. The average periodic pattern and frequency indicates that there is not a strict sampling at periodic time points (e.g., every hour on the hour sampling), and for a given time period (e.g., a day), there are a fixed number of measurement samples (we observed anywhere from 3 to 12 samples per day) at a deployment site.

To improve the detection times of our APD scheme in the traces, we evaluate using adaptive sampling instead of the average periodic sampling. In the case of adaptive sampling, once the plateau-detector enters the EI state, the sampling frequency at a site sf is increased to collect measurement samples at smaller inter-sampling times. Figure 11 shows the detection times when using the adaptive sampling with the sampling frequencies 2*sf, 4*sf, and 8*sf. To increase the number of samples available at higher frequencies at a deployment site, we use the values of the consecutive samples in the future. We do so with the assumption that they follow the same process of the anomaly event time series region just as the additional samples would have if they were adaptively sampled in reality. We can see that by using adaptive sampling measurement data, the APD scheme detection times reduce to the range of only a few hours versus the earlier observed ranges of several days. In addition, we can see that 4*sf and 8*sf cases would result in oversampling (i.e., measurement traffic consumes the network path bandwidth that could have been used by actual application traffic), and do not provide any further improvement to just using 2*sf. Nevertheless, the reduced detection times lessen the detection time tradeoff that is needed for reinforcement learning in our APD scheme.

TABLE I
TRACES DESCRIPTION

Trace ID	Source ↔ Destination	Time Range (Start - End)	Time Series Characteristics
1	psmsu02.aglt2.org ↔ psum02.aglt2.org	2009-10-9 15:03:19 - 2010-4-7 17:28:05	Persistent Decrease, Burst Decrease, Intermittent Dips
2	bwctl.ucsc.edu ↔ bwctl.atla.net.internet2.edu	2010-1-16 06:51:22 - 2010-4-7 20:36:05	Persistent Decrease, Persistent Increase, Intermittent Dips
3	bwctl.ucsc.edu ↔ bwctl.wash.net.internet2.edu	2010-1-16 08:50:36 - 2010-4-7 20:37:43	Persistent Decrease, Persistent Increase, Intermittent Bursts, Intermittent Dips
4	wtg248.otctest.psu.edu ↔ perfsonar.dragon.maxgigapop.net	2010-2-8 14:08:31 - 2010-4-7 21:25:57	Persistent Variations
5	chic-pt1.es.net ↔ anl-pt1.es.net	2009-7-2 20:04:41 - 2010-1-9 12:32:48	Persistent Increase, Persistent Decrease, Persistent Variations
6	nersc-pt1.es.net ↔ wash-pt1.es.net	2009-5-18 22:48:13 - 2010-1-9 16:46:47	Persistent Increase, Intermittent Bursts, Intermittent Dips
7	hous-pt1.es.net ↔ pnwg-pt1.es.net	2009-5-19 04:05:12 - 2010-4-7 13:39:31	Persistent Increase, Persistent Variations, Intermittent Dips
8	nettest.boulder.noaa.gov ↔ wtg248.otctest.psu.edu	2009-10-6 20:41:22 - 2010-4-7 21:27:05	Persistent Decrease, Persistent Increase, Intermittent Bursts, Intermittent Dips

TABLE II
ACCURACY EVALUATION RESULTS

Trace ID	$SPD_{s=2}$			$SPD_{s=3}$			$SPD_{s=4}$			$APD_{s=2...4}$		
No.	R_s	R_{f+}	R_{f-}	R_s	R_{f+}	R_{f-}	R_s	R_{f+}	R_{f-}	R_s	R_{f+}	R_{f-}
1	1	0	**2**	1	0	1	1	0	0	1	0	0
2	1	0	**1.5**	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**	0	1	0	0
3	1	0	0	**0.67**	**0.33**	**0.33**	1	0	0	1	0	0
4	**0.5**	**0.5**	**5**	1	0	0	1	0	0	1	0	0
5	1	0	**0.5**	1	0	0	**0.5**	**0.5**	0	1	0	0
6	**0**	0	**3**	1	0	**2**	1	0	0	1	0	0
7	1	0	**0.5**	**0.5**	**0.5**	0	**0.5**	**0.5**	0	1	0	0
8	1	0	**0.5**	1	0	0	1	0	0	1	0	0

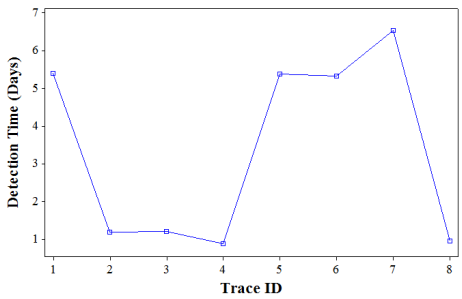


Fig. 10. Effect of average periodic sampling on anomaly detection times

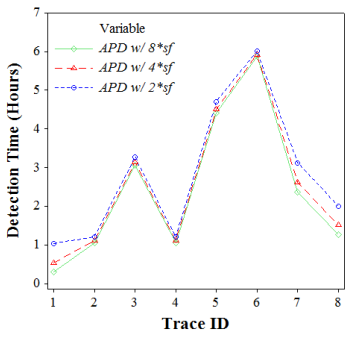


Fig. 11. Effect of adaptive sampling on anomaly detection times

D. Scalability Evaluation

Lastly, we evalute the scalability of the APD scheme if used for notifying anomalies in large-scale measurement topologies comprising of several hundred of network paths. For this, we calculate the average analysis time per-site to sequentially detect anomalies on over 480 perfSONAR monitored paths whose measurement data we queried from 65 sites using our OnTimeDetect tool. We define analysis time for a measurement archive at a site as the sum of the times taken for perfSONAR web-service processing, measurement archive transfer and run time of the APD scheme on the measurement archive. Figure 12 shows the average analysis time per-site for sequential queries spanning multi-timescale resolutions shown on the x-axis. We remark that the average analysis times shown have been calculated from 75 query iterations spanning several days. We can observe that the analysis times are on the order of tens of seconds. Upon further analysis, we found that some sites have analysis times that are less than a second, whereas some sites have analysis times on the order of several seconds. Based on this observation, we experimented with a parallel query mechanism where the OnTimeDetect concurrently queries all the sites in the site-list. The speed up results in the average analysis time per-site for the parallel query compared to the sequential query from 75 iterations are shown in Figure 13. The speedup value is calculated as a percentage ratio of the sequential query analysis time over the

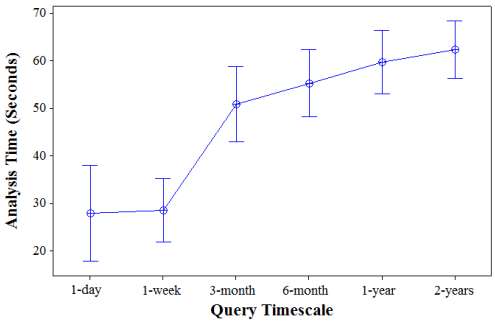


Fig. 12. Per-site analysis times for sequential queries spanning multi-timescale resolutions

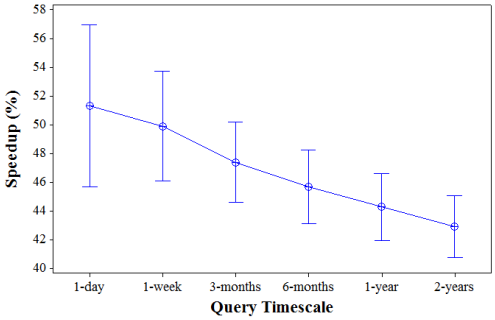


Fig. 13. Speedup in per-site analysis times for parallel queries spanning multi-timescale resolutions

parallel query analysis time. We can see that the parallel-query mechanism in the OnTimeDetect tool can speedup the average analysis time per-site by (40 - 60)%, and inturn improve the OnTimeDetect tool user experience during online drill-down analysis.

VI. Conclusion

In this paper, we presented a dynamically adaptive plateau-detector and its implementation in our "OnTimeDetect" tool to notify network anomaly events that are of interest to network operators and end-users in the data-intensive scientific communities. We leveraged the vast measurement data archives available via perfSONAR web-services at several worldwide sites, and also synthetic measurement traces with anomaly events to develop our adaptive plateau-detector.

Through our performance evaluations, we showed that our adaptive plateau-detector is robust and accurate in anomaly notifications. We also showed that it is possible to minimize anomaly detection times in the perfSONAR deployments by using adaptive sampling instead of the currently employed average periodic sampling. Further, we showed that it is possible to substantially speedup anomaly analysis times when dealing with large number of paths using a parallel-query mechanism instead of a sequential-query mechanism.

Our future work is to compare and contrast our adaptive plateau-detector with other advanced anomaly detection schemes considering archives of additional network health metrics available in the perfSONAR deployments.

References

[1] Teragrid Project - http://www.teragrid.org

[2] The Large Hadron Collider (LHC) Project - http://lhc.web.cern.ch/lhc

[3] A. Hanemann, J. Boote, E. Boyd, J. Durand, L. Kudarimoti, R. Lapacz, M. Swany, S. Trocha, J. Zurawski, "PerfSONAR: A Service Oriented Architecture for Multi-Domain Network Monitoring", *Proc. of Service Oriented Computing, Springer Verlag, LNCS 3826*, pp. 241-254, 2005. (http://www.perfsonar.net)

[4] P. Calyam, A. Kalash, R. Gopalan, S. Gopalan, A. Krishnamurthy, "RICE: A Reliable and Efficient Remote Instrumentation Collaboration Environment", *Advances in Multimedia Journal*, 2008.

[5] J. Zurawski, M. Swany, D. Gunter, "Scalable Framework for Representation and Exchange of Network Measurements", *Proc. of IEEE TRIDENTCOM*, 2006.

[6] C. Guok, D. Robertson, M. Thompson, J. Lee, B. Tierney, W. Johnston, "Intra and Interdomain Circuit Provisioning Using the OSCARS Reservation System", *Proc. of IEEE/ICST Conference on Broadband Communications, Networks, and Systems*, 2006.

[7] J. Allen, "Driving by the Rear-View Mirror: Managing a Network with Cricket", *Proc. of USENIX Network Administration Conference*, 1999.

[8] W. Matthews, L. Cottrell, "The PingER Project: Active Internet Performance Monitoring for the HENP Community", *IEEE Communications Magazine on Network Traffic Measurements and Experiments*, 2000.

[9] A. Lakhina, M. Crovella, C. Diot, "Diagnosing Network-Wide Traffic Anomalies", *Proc. of ACM SIGCOMM*, 2004.

[10] P. Barford, J. Line, D. Plonka, A. Ron, "A Signal Analysis of Network Traffic Anomalies", *Prof. of ACM SIGCOMM Internet Measurement Workshop*, 2002.

[11] A. Soule, K. Salamtian, N. Taft, "Combining Filtering and Statistical Methods for Anomaly Detection", *Proc. of Internet Measurement Conference*, 2005.

[12] M. Thottan, G. Liu, C. Ji, "Anomaly Detection Approaches for Communication Networks", *Book Chapter in Algorithms for Next Generation Networks*, Springer, 2010.

[13] A. McGregor, H-W. Braoun, "Automated Event Detection for Active Measurement Systems", *Proc. of Passive and Active Measurement Workshop*, 2001.

[14] C. Logg, L. Cottrell, "Experiences in Traceroute and Available Bandwidth Change Analysis", *Proc. of ACM SIGCOMM Network Troubleshooting Workshop*, 2004.

[15] L. Cottrell, M. Chhaparia, F. Haro, F. Nazir, M. Sandford, "Evaluation of Techniques to Detect Significant Network Performance Problems using End-to-end Active Network Measurements", *Proc. of IEEE/IFIP NOMS*, 2006.

[16] F. Feather, R. Maxion, "Fault Detection in an Ethernet Network using Anomaly Signature Matching", *Proc. of ACM SIGCOMM*, 1993.

[17] H. Hajji, "Statistical Analysis of Network Traffic for Adaptive Faults Detection", *IEEE Transactions on Neural Networks*, 2005.

[18] M. Thottan, C. Ji, "Anomaly Detection in IP Networks", *IEEE Transactions on Signal Processing*, 2003.

[19] D. Gunter, B. Tierney, A. Brown, M. Swany, J. Bresnahan, J. Schopf, "Log Summarization and Anomaly Detection for Troubleshooting Distributed Systems", *Proc. of IEEE/ACM Grid Computing*, pp. 226-234, 2007.

[20] L. Yang, C. Liu, J. Schopf, I. Foster, "Anomaly Detection and Diagnosis in Grid Environments", *Proc. of ACM/IEEE Supercomputing*, 2007.

[21] A. Tirumala, L. Cottrell, T. Dunigan, "Measuring End-To-End Bandwidth with Iperf Using Web100", *Proc. of Passive and Active Measurement Workshop*, 2003.

[22] Thrulay: Network Capacity Tester - http://shlang.com/thrulay

[23] Nuttcp: TCP/UDP Network Testing Tool - http://wcisd.hpc.mil/nuttcp

[24] P. Calyam, A. Devulapalli, "Modeling of Multi-resolution Active Network Measurement Time-series", *Proc. of IEEE Workshop on Network Measurements*, 2008.

[25] P. Calyam, D. Krymskiy,M. Sridharan, P. Schopis, "Active and Passive Measurements on Campus, Regional and National Network Backbone Paths", *Proc. of IEEE ICCCN*, 2005.

[26] P. Calyam, D. Krymskiy,M. Sridharan, P. Schopis, "TBI:End-To-End Network Performance Measurement Testbed For Empirical Bottleneck Detection", *Proc. of IEEE ICCCN*, 2005.

[27] M. Wheelwright, R. Hyndman, "Forecasting: Methods and Applications (3rd edition)", *Wiley Publication ISBN No. 0471532339*, 1998.