A Distributed and Reliable Platform for Adaptive Anomaly Detection in IP Networks

L. Lawrence Ho, Christopher J. Macey, and Ronald Hiller

Communications Sciences Research Laboratory
Bell Laboratories, Lucent Technologies
101 Crawfords Corners Road
Holmdel, NJ 07733, USA
{llho, macey, ronbo}@bell-labs.com

Abstract. Algorithms for anomaly detection in IP networks have been developed and a real-time distributed platform for anomaly detection has been implemented. These algorithms automatically and adaptively detect "soft" network faults (performance degradations) in IP networks. These algorithms are implemented as a reliable and fully distributed real-time software platform called NSAD (Network/Service Anomaly Detector). IP NSAD has the following novel features. First, it provides a flexible platform upon which pre-constructed components can be mixed/matched and distributed (to different machines) to form a wide range of application specific and fully distributed anomaly detectors. Second, anomaly detection is performed on raw network observables (e.g., performance data such as MIB2 and RMON1/2 variables) *and* algebraic functions of the observables (objective functions), making NSAD an objective driven anomaly detection system of wide detection range and high detection sensitivity. Third, controlled testing demonstrates that NSAD is capable of detecting network anomalies reliably in IP networks.

1 Introduction and Algorithmic/System Design

Anomaly detection is concerned with detecting "soft" network faults (e.g., performance degradation) in network and their services, which in turn enables "proactive" containment and correction of network/service faults for reliable networking [1,2]. Since anomalies are signatures of network/service exceptions and are preludes to service level failures, being able to detect them reliably and quickly enables network failures to be anticipated (hence "proactive") [1–5]. This implies that fault correction and containment can be applied in a timely manner to manage the performance and optimize the reliability of networks and their services.

With the explosive growth of heterogeneous data networks (the Internet and intranets) in the past decade, there has been a corresponding leap in the types, frequencies, and severity of network/service faults and outages [6]. Some examples of these network faults include network mis-configuration, hardware problems (e.g., line card failure), networking software exceptions (e.g., "bugs" in routing software, exceptions in routing table updates), "bugs" in network-enabled applications (e.g., video streaming software), traffic anomalies (e.g., packet storms), and so on [7–10]. These anomalies and faults, if not contained and corrected in time, will degrade network quality-of-service (QoS) parameters such as delay, jitter, and packet drop/error rate, which in turn lead to network/service degradations and ultimately failures. As modern data networks and their support software become more complex (in physical/logical topologies and software complexity), the number of potential network/service points of failures are increasing exponentially, their faults correspondingly less predictable, and detecting them increasingly difficult.

Recently, there has been another trend in networking—increasing demand on network QoS and reliability [11]. As electronic commerce (E-commerce) and real-time applications (e.g., video and audio based applications) become more prevalent, guaranteeing QoS and reliability is becoming increasingly important [12,13]. This in turn demands that network/service faults be detected and identified reliably and quickly for rapid fault alert, containment, and correction [1–5].

Together, these two trends have fueled the recent surge in research concerning network fault detection and performance management [14–17]. These challenges motivated our previous research in anomaly detection in non-IP based networking environment [1,2], and our current work in IP based anomaly detection.

To detect network/service anomalies and faults reliably and in real-time, two set of design factors are relevant, one set is algorithmic, the other system-related [1,2]. For network/service anomaly detection, algorithms should be:

- *Adaptive to the performance fluctuation and the evolution of networks and their services*
- *Performing detection in real-time*
- *A reliable (self-recovering) real-time system*
- *A flexible and distributed system*

It will be explained in detail that most fault detection systems to date do not achieve all of the above design goals, including algorithms and systems developed by other researchers and us [1–5,24–27]. The current IP network and service anomaly detection (IP NSAD) system reported in this paper fulfills these design requirements.

Traditionally, research and development in network fault management emphasize the detection and processing of network and element failures in the form of alarms [18]. For example, most network elements such as routers and switches are equipped with software agents (e.g., SNMP agents) that can emit traps/alarms upon detecting exceptional states such as a router interface being "down," or the utilization of a network segment exceeding a predefined threshold (e.g., a constant 80% threshold). Event correlation deals with inferring the root cause(s) of network faults from an avalanche of network alarms. Recent research and deployment testify to its usefulness in monitoring and controlling networks [19]. In network fault management, the past few years have also witnessed much progress in path failure detection (including break faults) [20–22], billing fraud detection in telecommunication voice networks

[23], anomaly detection in Ethernet [24,25], anomaly and performance change detection in small networks [26,27], and network alarm correlation [28–30]. With the advent and the explosive growth of the global Internet and the electronic commerce environments, adaptive/automatic network and service anomaly detection in IP wide area data networks and IP based E-commerce infrastructures (e.g., web based transaction infrastructures) is fast gaining critical research and practical importance [5]. In these cases, being able to proactively detect performance degradations (termed "soft" faults [1–5] as opposed to the "hard" alarms/failures of networks and their elements [18,19,28–30]) in networks and their services is becoming crucial for speedy fault recovery, and for preventing the onset of network/service failures.

Most of the developed algorithms and their implementations do not fully conform to the above algorithmic and system-related design requirements. For example, most fault detection systems built to date [1–3] are inflexible (meaning they are application specific) and unreliable. Therefore, they are not general-purpose anomaly detection platforms. Given the ever-accelerating development/deployment pace of new type of network services and new network-enabled applications, new types of application specific anomaly detectors are constantly required. Therefore, the lack of flexibility and generality in current fault detection systems may hinder their usefulness. On the algorithmic side, not all the algorithms designed [1–5,25–27] have been subject to controlled testing with realistic anomaly injection, which could have validated their effectiveness (e.g., detection hit rates, false positive rates, and so on). Instead, they were tested and optimized against historical network and service data and their corresponding historical fault information (usually in the form of trouble tickets) [1–5,25–27]. These fault information sources are notoriously hard to collect and unreliable. Even if available, they are not specific enough to enable vigorous testing and optimization. In our previous work [1,2], this problem was addressed (in a best effort way) by collecting very low-level, highly specific, and real-time trouble ticketing information for algorithmic verification. Nevertheless, the drawbacks were apparent [1,2].

Motivated by these algorithmic and system drawbacks in previous work, we recently designed a set of IP network/service anomaly detection algorithms, implemented a real-time distributed anomaly detection platform, and vigorously tested them. This platform is called the IP network and service anomaly detector (IP NSAD). Specifically, IP NSAD is:

- *Capable of adaptive and automated detection of anomalies in IP networks and their services in real time. NSAD uses as inputs SNMP based standard network/service observables such as MIB2 and RMON1/2 variables [31],*
- *Capable of objective driven and highly effective anomaly detection. In this case, anomaly detection is performed on performance objective functions (as opposed to raw network observables) which are algebraic functions of standard based network/service observables (MIB2, RMON1/2, etc.),*
- *A flexible platform on which pre-constructed components (such as network samplers, data filters, rule generators, and anomaly detectors, etc.) can be*

mixed and matched (through an intuitive graphical user interface, GUI) for building application specific anomaly detectors,

- *A fully distributed system in which system components can be distributed to different platforms, and*
- *A reliable real-time system in which components are dynamically self-recovering.*

This paper is structured as follows. An overview of the anomaly detection algorithms is presented in Section 2. Section 3 details the system architecture and implementation of the IP NSAD platform. Section 4 summarizes test results of IP NSAD. Finally, Section 5 provides a summary.

2 Algorithms for IP Network/Service Anomaly Detection

This section provides a summary of the anomaly detection algorithms, which is an extension of our previous work [1,2]. The anomaly detection algorithms can be grouped into three categories: (1) sampling and filtering algorithms, (2) statistical analysis and rule generation algorithms, and (3) anomaly detection algorithm.

A network anomaly detector is a real-time program that adaptively analyzes performance data of managed networks to detect "abnormal" changes (relative to historical baselines or "expected" behavior) in traffic and performance. These abnormalities are signatures of soft and hard faults. For IP networks, an anomaly detector analyzes SNMP based network observables (i.e., MIB2 and RMON1/2 variables [31]) to identify fault location, raise alarms, and generate control signals upon detection of performance anomalies and network faults.

The three steps of adaptive network anomaly detection are:

1. Network data sampling and filtering
   For best effort correction of stochastic effects in network monitoring and for handling missing data.

   MIB2 or RMON1/2 counter (or gauge) values [31] generated by network elements (e.g., routers, switches, etc.) or probes (RMON probes) are retrieved periodically (in a best effort way with predefined time intervals) through SNMP. This network performance sampling uses a "overstepping" algorithm which forces the time stamps of the return SNMP PDUs to be slightly (bounded) later than the predefined (or theoretical) retrieval time stamps, taking into account the stochastic network delays on the SNMP PDUs. This in effect minimizes error interpolation. The time stamps of the resulting counter values are closely aligned, but not exactly, to a regular schedule. This misalignment is due to stochastic fluctuations in network delay, which affect the arrival times (with respect to the regular schedule) of the SNMP PDUs. These counter values are interpolated to form a set of exactly regular time series. Interpolation is used to synthesize missing counter values for missing regions whose sizes do not exceed predefined bounds.

First and higher order derivatives of counters can be computed directly from their regular time series.

2. Temporal-based performance thresholds
For predicted or "expected" baselines and their fluctuations of network performances.

By exploiting the temporal performance regularities of networks, performance baselines and thresholds of IP networks can be built for their MIB2 or RMON1/2 counters. These baselines and thresholds can be classified into four classes: weekdays, Saturdays, Sundays, and holidays. Historical data of service classes are used to construct these adaptive thresholds for each monitored MIB2 or RMON1/2 counter.

For each of the four threshold groups, a set of adaptive thresholds are built to predict the expected performance of network services on weekdays, Saturdays, Sundays, and holidays, respectively. Each set of dynamic thresholds (upper and lower thresholds) is composed of a predicted baseline $\tilde{I}_s\left(T_{n,s}\right)$ and tolerance

$$\text{upper\_threshold} = \tilde{I}_s\left(T_{n,s}\right) + 2\tilde{\sigma}_s\left(T_{n,s}\right)\Big|_{\substack{wkdys\\ sats\\ suns\\ holiday}} \tag{1}$$

$$\text{baseline} = \tilde{I}_s\left(T_{n,s}\right)\Big|_{\substack{wkdys\\ sats\\ suns\\ holiday}} \tag{2}$$

$$\text{lower\_threshold} = \tilde{I}_s\left(T_{n,s}\right) - 2\tilde{\sigma}_s\left(T_{n,s}\right)\Big|_{\substack{wkdys\\ sats\\ suns\\ holiday}} \tag{3}$$

$\tilde{\sigma}_s\left(T_{n,s}\right)$ (note: "~" denotes "predicted") as follows

The baseline $\tilde{I}_s\left(T_{n,s}\right)$ and tolerance $\tilde{\sigma}_s\left(T_{n,s}\right)$ are computed from historical MIB2 or RMON1/2 data (i.e., counters) through one-dimensional time series analysis and are classified into the "weekday", "Saturday", "Sunday", and "holiday" classes [1,2]. The $\tilde{I}_s\left(T_{n,s}\right)$s represent the predicted "average" counters, while the $\tilde{\sigma}_s\left(T_{n,s}\right)$s represent the predicted "average" fluctuations of the corresponding counters. Both $\tilde{I}_s\left(T_{n,s}\right)$s and $\tilde{\sigma}_s\left(T_{n,s}\right)$ are updated periodically to account for the evolution in network traffic.

The baseline $\tilde{I}_s\left(T_{n,s}\right)$ and tolerance $\tilde{\sigma}_s\left(T_{n,s}\right)$ can also be computed for objective functions, which are algebraic functions of raw network observables. These performance based objective functions provide sensitive measures of application specific performance. For example, an objective function can measure the traffic inbalance between the inbound traffic flowing into a network region and the outbound traffic. This objective function is

$$f_{obj}(t) = \frac{d}{dt}\left[\sum_{\text{interfaces}} (ifOctetIn + ifOctetOut)\right], \tag{4}$$

where *ifOctetIn* and *ifOctetOut* are MIB2 counters that register byte counts into an out of an router interface. Finally, the summation is performed over all interfaces related to the network region. The resulting objective function (Equation 4) enables detection of traffic inbalance anomalies.

3. Anomaly detection
   For comparing expected baselines/fluctuations with current network states to measure departures from predicted performances, which are signatures of anomalies.

Expected performance of IP networks are predicted through the above thresholds, and deviations (in both magnitude and duration, as defined by a set of fault criteria) from the expected are indications of network/service anomalies [1,2].

In anomaly detection, an alarm is raised that signaling the onset of a network/service anomaly if (1) the measured (in real-time) $I_{s,measured}(T_{n,s})$ at time $T_{n,s}$ deviates from the thresholds by more than a from the predicted baseline, and (2) the previous condition persists and for more than $T_{persist}$, i.e.,

$$\left|\sum_{T}^{T_{persist}} E_s(T_{n,s})\right| \geq \sum_{T}^{T_{persist}} 2\tilde{\sigma}_s(T_{n,s}), \tag{5}$$

where

$$\text{if } I_{s,measured}(T_{n,s}) \leq \left[\tilde{I}_s(T_{n,s}) - 2\tilde{\sigma}_s(T_{n,s})\right] - a\tilde{I}_s(T_{n,s}):$$
$$E_s(T_{n,s}) = I_{s,measured}(T_{n,s}) - \left[\tilde{I}_s(T_{n,s}) - 2\tilde{\sigma}_s(T_{n,s})\right] - a\tilde{I}_s(T_{n,s}) \tag{6}$$
$$\text{if } I_{s,measured}(T_{n,s}) \geq \left[\tilde{I}_s(T_{n,s}) + 2\tilde{\sigma}_s(T_{n,s})\right] + a\tilde{I}_s(T_{n,s}):$$
$$E_s(T_{n,s}) = I_{s,measured}(T_{n,s}) - \left[\tilde{I}_s(T_{n,s}) + 2\tilde{\sigma}_s(T_{n,s})\right] + a\tilde{I}_s(T_{n,s}). \tag{7}$$

The choice of the parameters in the above criteria (a and $T_{persist}$) are determined experimentally. Finally, the detected anomaly is written to a diagnosis log that identifies the "guilty" elements and the possible cause(s) of anomaly. This information is presented to network operators through a graphic user interface (GUI).

3 System Architecture and Implementation of NSAD

In this section, the system architecture of the IP NSAD is presented. The corresponding implementation is also detailed.

The overall system architecture of the IP NSAD platform is shown in Figure 1, which presents the various system components that make up an anomaly detector. As will be explained in detail, these components are responsible for gathering network performance data from network elements or probes (samplers), numerically and arithmetically processing the gathered data (filters), statistically analyzing the gathered data (rule generators), and finally performing real-time anomaly detection (detectors). These components are monitored and controlled by a centralized master controller, which can distribute the above system components to different machines connected by LAN(s) or WAN(s). Thereby, computationally and network-bandwidth intensive operations (such as rule generation of predicted baselines and thresholds, and network sampling, respectively) can be distributed to a set of processors and network segments for load balancing and for enhancing reliability. Further, local traffic reduction (by distributing some of the filters) reduces the bandwidth required for subsections of the system and reduces the managed scope. This is especially important when configuring data collection and anomaly detection for a wide area network, in which firewalls or other policy managed domains may need to be traversed. Currently, platforms supported by the core NSAD components and their master controller include Sun's Solaris 7 and Microsoft's Windows NT 4.0. Human interaction with the NSAD platform is conducted through a graphical user interface (GUI), which enables system construction (i.e., building anomaly detectors from system components), configuration, and advanced performance/anomaly visualization of IP networks.

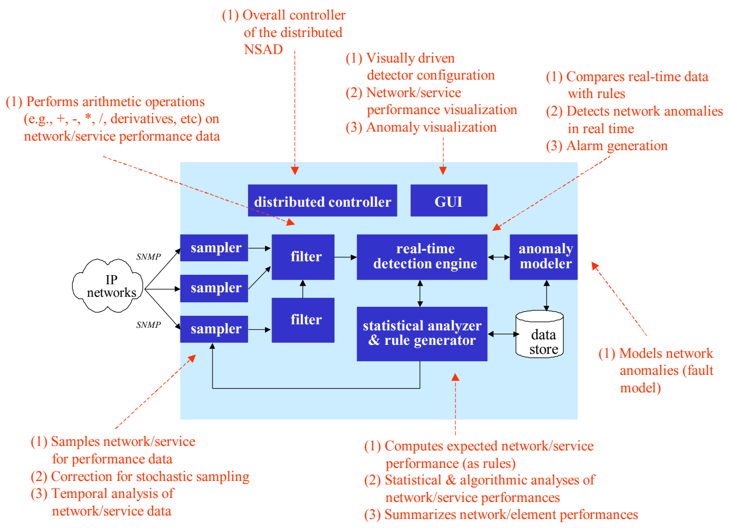
(1) Overall controller of the distributed NSAD
(1) Visually driven detector configuration
(2) Network/service performance visualization
(3) Anomaly visualization
(1) Compares real-time data with rules
(2) Detects network anomalies in real time
(3) Alarm generation
(1) Performs arithmetic operations (e.g., +, -, *, /, derivatives, etc) on network/service performance data
distributed controller
GUI
IP networks
SNMP
sampler
sampler
sampler
SNMP
filter
filter
real-time detection engine
anomaly modeler
statistical analyzer & rule generator
data store
(1) Models network anomalies (fault model)
(1) Samples network/service for performance data
(2) Correction for stochastic sampling
(3) Temporal analysis of network/service data
(1) Computes expected network/service performance (as rules)
(2) Statistical & algorithmic analyses of network/service performances
(3) Summarizes network/element performances

Fig. 1. Architecture of NSAD, showing the system components of a single anomaly detector.

In real applications, multiple anomaly detectors can be controlled and distributed by a master controller. For example, in an E-Commerce infrastructure composed of servers, routers, and Fast Ethernet switches, one detector may be used for web server anomaly detection (server overload, denial of service, etc.), while others may be configured to detect traffic anomalies by sampling router and switch performance data. This deployment of interrelated detectors in multiple points of a network is a key to effective performance monitoring and fault detection.

The architecture of NSADS is implementation-independent. It relies on a small number of behavioral and communication conventions. It consists of a number of simple system components (Figure 1) which are interconnected by communications links and provide time-stamped values (according to a fixed schedule) for further processing. The current implementation is in C for maximal efficiency. A Java version is planned and Inferno, CORBA and even hardware implementations are envisioned.

A detector instance is a configuration of an arbitrary number of these components, connected in a directed a-cyclic graph. The instance computes an objective function of the input observables, which can then be statistically modeled (Figure 1). Instances are configured from templates that describe generic versions of each supported detector type.

The user selects a supported type of fault detector interactively, then instantiates actual detectors with a minimal specification of bindings to the target network. Once instantiated and launched the detector is robust and generally requires no further user intervention, even in the presence of network failures. Adaptation to network growth is accommodated with simple editing of existing configurations. Scalability is provided by redistribution of component instances across the network.

The core system components of the NSAD platform include the (1) samplers, (2) filters, (3) rule generators, (4) detectors, (5) anomaly modeler, (6) master controller, and (7) GUI. Their functions are as follows.

- *Sampler*
  Samplers are data retrieval units that periodically poll network elements (including servers) and monitoring probes (e.g., RMON probes) for network/service performance data. Currently, they support (1) SNMP based (variable based and table based) and (2) flat file based retrieval. Therefore, MIB2 variables, RMON1/2 variables, and generic data (e.g., web server logs) can be retrieved periodically or on demand. Effectively, this module implements the first part of the anomaly detection algorithms outlined in Section 2. For periodic retrieval (i.e., with a fixed frequency), SNMP PDUs are generated on a fixed schedule (for example every 10 seconds starting at 0:00:00 GMT). The return SNMP PDUs (which carry the performance data) are not spaced at 10 seconds

interval, owing to the stochastic delay of the network(s) involved. Therefore, a "overstepping" algorithm is used to guarantee that every return PDU is displaced (time-wise) slightly later than the ideal 10 second time grid, without divergence (running over into the next time bin). Interpolation is used to realign the data back to a 10 second spaced (regular) time series. In addition, algorithms are provided to handle missing data. This sampling schedule (interval and start time) generate the clock for the entire system.

- *Filter*
  Filters perform numerical and arithmetic computation on sampled network data. Supported numerical operations include derivative calculations. Supported arithmetic operations include summation, difference, multiplication, and division. Using these filters, network observables (such as individual MIB2 or RMON1/2 variables) can be combined into performance objective functions for more effective anomaly detection. For example, traffic inbalance between inbound and outbound octet rates of a router can be measured and computed in real time.
- *Rule generator*
  Rule generators are the modules that implement the statistical analysis algorithms (second part of the algorithms outlined in Section 2) for generating predicted baselines and fluctuations of the performance variables or their objective functions. They analyze the stored historical performance data or their composite objective functions (filter outputs) of networks periodically, and compute the predicted baselines of these. Predicted fluctuations of the corresponding baselines are then computed. Rule generators are executed periodically, and infrequently compared to the frequency of real-time sampling. Typically, they are executed a few times per day (in our present system, every 12 hours). This means baselines and fluctuation rules for variables or objective functions are predicted for 12 hours periods on an ongoing basis (12 hours ahead of their use).
- *Detector*
  Detectors are executed in real time to detect network and service anomalies. They compare the predicted rules (in terms of baselines and their fluctuations) against their corresponding real time network data (or their objective functions) with respect to a set of anomaly criteria. These criteria are in the form of the algorithms outlines in Section 2 (part 3 of the set). Violations of the criteria are interpreted as anomalies. Degrees of violation are mapped to conditional probabilities, i.e., how likely the detected violations are real anomalies.
- *Graphic User Interface (GUI)*
  The GUI provides users the following functions. First, it provides advanced visualization capabilities for users to graphically view raw network performance, their statistical summaries, and network anomalies. Second, it enables users to graphically build anomaly detectors from components. Third, it allows the user to set system parameters (e.g., the machines on which to execute specific system components and the data sampling schedule). A screen shot of the NSAD GUI is shown in Figure 2, highlighting the detector construction and anomaly visualization functions. The GUI is written in Java, and is therefore platform independent. It can be remotely invoked through a web browser or executed as a standalone Java application (Figure 2).

- *Controller*
  A controller runs on each machine that will execute anomaly detector components. Typically, it is automatically initiated on machine boot. It launches and monitors all other components except rule generation. Failed components are re-spawned as necessary. Controllers communicate with one another on a well-known TCP port and mediate all communication between the GUI and other components.

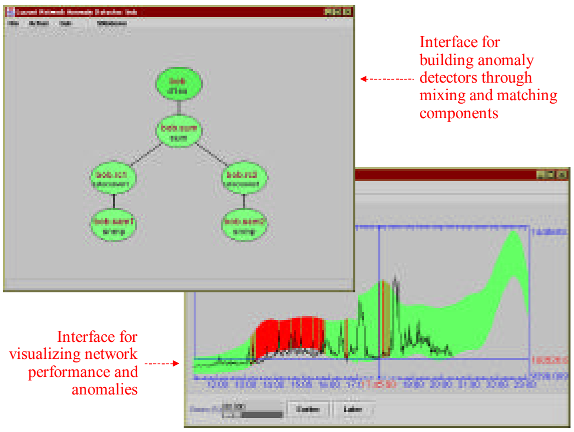
Interface for building anomaly detectors through mixing and matching components
Interface for visualizing network performance and anomalies

Fig. 2. A snapshot of the NSAD GUI.

The primary recovery mechanism provided by NSAD handles the loss of inter-component connections and file system unavailability. If a component detects the loss of a connection to any of its downstream or upstream connections, it simply terminates. This causes a cascade of terminations through the entire configuration. Even components hosted on other machines are cleanly shutdown so recovery can be initiated. Since the topmost component in any configuration is connected to the local controller, its termination initiates the recovery. If this connection is lost, the controller reinitiates the configuration using an exponential backoff strategy for repeated failures. If the controller fails, it is relaunched and then relaunches the components.

The combination of configuration files and controllers provides a mechanism for partitioning recovery. Any portion of a detector can be converted into a separate configuration. The topmost component of the new configuration will only terminate if it loses a downstream connection or the connection to the controller. This can be used to limit the downstream propagation of failures. When the upstream components have recovered from a non-local failure, they will reconnect to the subsystem.

If the local file system becomes unavailable, local components are prevented from recording their values. Instead, they queue values in available memory and log requests for operator intervention. If the rule bases are unavailable, the detector logs the problem and becomes quiescent.

Scalability is provided through the distributed nature of the detector and the use of remote aggregation to limit the quantity of data which must be moved upstream over any single link.

4 An Application and Preliminary Test Results

This section provides an example of IP anomaly detection, and a summary of some initial test results performed in our specially developed controlled testbed, where anomalies are injected to test the algorithmic performance of the IP NSAD.

Construction of anomaly detector(s) can be done intuitively through the GUI by a series of drag-and-drop steps, and the filling in of some system related and functional parameters (e.g., IP addresses for data polling and polling frequencies, respectively). Figure 3 shows an example of constructing an anomaly detector through the NSAD GUI. In this case, an anomaly detector is constructed to monitor the overall (summation of inbound and outbound) traffic (byte count) rate tranversing a router interface. The objective function being monitored is rate of the summation of the inbound-outbound interface octet counts, i.e.,

$$f_{obj}(t) = \frac{d}{dt}(ifOctetIn + ifOctetOut), \tag{8}$$

which is a sensitive measure of the traffic and load conditions on a subnetwork, as well as the health of a router interface.

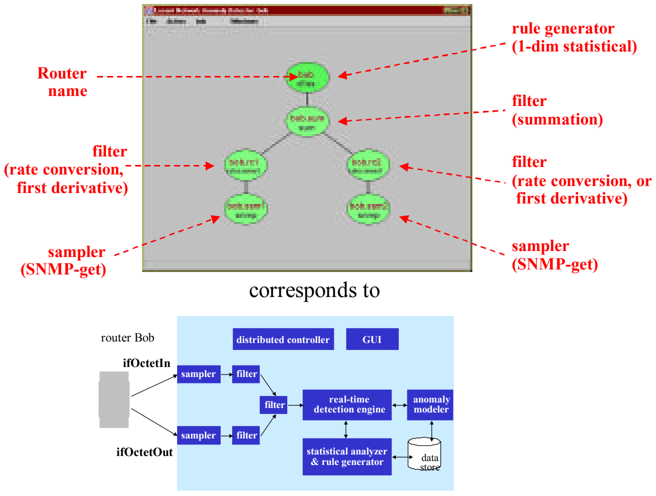

Fig. 3. A snapshot of the NSAD GUI, showing the detector construction .panel, and the corresponding anomaly detector.

In this case, the rule generator generates the dynamic baselines and fluctuations of the objective function (Equation 7). Figure 4 shows traffic anomalies detected by the detector.

To enable controlled testing of IP NSAD, we have constructed a special-purpose network testbed. This generates realistic background traffic, and can inject anomalies of different classes (e.g., web server overload, traffic storm, etc.) [32]. Preliminary results indicate that NSAD is capable of highly accurate anomaly detection. For example, in anomaly detection of web servers, the average hit rate exceeds 98%, with 2% of false positives.

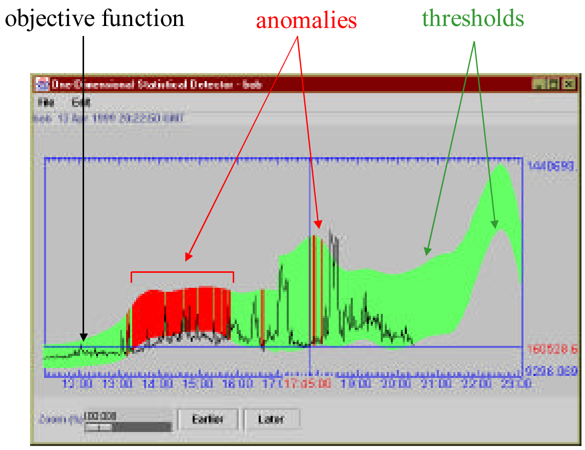

Fig. 4. A snapshot of the NSAD GUI, showing the visualization panel.

5 Conclusions

We have developed a general platform for IP anomaly detection. This IP NSAD

Ongoing and future work IP NSAD include:

- Vigorous controlled testing with realistic anomaly injections,
- Implementation of additional components including new filters, samplers, and rule generators, and
- System enhancements through re-implementing selected components with Java and CORBA.

References

1. Ho, L.L., Cavuto, D.J., Papavassiliou, S., Hasan, M.Z., Feather, F.E., Zawadzki, A.G., "Adaptive Network/Service Fault Detection in Transaction-Oriented Wide Area Networks," *Proceedings of the Sixth IFIP/IEEE*

International Symposium on Integrated Network Management (IM'99), Edt. M. Sloman, S. Mazumdar, and E. Lupu, (IEEE Press), to appear in May 1999.

2. Ho, L.L., Cavuto, D.J., Papavassiliou, S., Zawadzki, A.G., "Adaptive and Automated Detection of Network/Service Anomalies in Wide Area Networks," *Journal of Network and Systems Management*, to appear in 1999.
3. Thottan, M., Ji, C., "Fault Prediction at the Network Layer using Intelligent Agents," *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management (IM'99)*, Edt. M. Sloman, S. Mazumdar, and E. Lupu, (IEEE Press), to appear in May 1999.
4. Hood, C. and Ji, C., `` Intelligent Processing Agents for Network Fault Detection", *IEEE Internet Computing*, Vol. 2, No. 2, March/April 1998
5. Hellerstein, J.L., Zhang, F., Shahabuddin, P., "An Approach to Predictive Detection for Service Management," *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management (IM'99)*, Edt. M. Sloman, S. Mazumdar, and E. Lupu (IEEE Press), to appear in May 1999.
6. Huberman, B.A., Lukose, R.M., "Social Dilemmas and Internet Congestion," *Science*, Vol. 277, p. 535, July 1997.
7. Held, G., *LAN Testing and Troubleshooting: Reliability Tuning Techniques*, John Wiley & Sons, 1996.
8. Ballew, S.M., *Managing IP Networks*, O'Reilly & Associates, 1997.
9. Miller, M.A., *Troubleshooting Internetworks*, M&T Publishing, 1991.
10. Espinosa, R., Tripod, M., Tomic, S., *Cisco Router Configuration & Troubleshooting*, New Riders, 1998.
11. Kumar, V.P., Lakshman, T.V., Stiliadis, D., "Beyond Best-Effort: Gigabit Routers for Tomorrow's Internet," *IEEE Communications Magazine*, V36(5), p152, May 1998
12. White, P.P., "RSVP and Integrated Services in the Internet: A Tutorial," *IEEE Communications Magazine*, V35(5), p100, 1997.
13. Reininger, D., " A Dynamic Quality of Service Framework for Video in Broadband Networks," *IEEE Network*, V12(6), p22, 1998.
14. Lazar, A.A., Wang, W., Deng, R., "Models and Algorithms for Network Fault Detection and Identification: A Review," *ICC Singapore*, Nov. 1992.
15. Parulkar, G., Schmidt, D., Kraemer, E., Turner, J., Kantawala, A., "An Architecture for Monitoring, Visualization, and Control of Gigabit Networks," *IEEE Networks*, p.34, Sept/Oct, 1997.

16 Katzela, I. Schwartz, M., "Schemes for Fault Identification in Communication Networks," *IEEE/ACM Trans. Networking*, Vol. 3(6), p.753, Dec, 1995.

17. Aidarous, S. (Edt.), Plevyak (Edt.), "Telecommunications Network Management: Technologies and Implementations," *IEEE Series on Network Management*, (IEEE Press, 1998).
18. Aidarous, S. (Edt.), Plevyak (Edt.), "Telecommunications Network Management into the 21st Century: Techniques, Standards, Technologies, and Applications," (IEEE Press, 1994).
19. Yemini, S., Kliger, S., Mozes, E., Yemini, Y., Ohsie, D., "High Speed and Robust Event Corrrelation," *IEEE Communication Magazine*, May 1996.

20. Wang, C., Schwartz, M., "Fault Diagnosis of Network Connectivity Problems by Probabilistic Reasoning," *Network Management and Control Volume Two* (Ed. Frisch, I.T., Malek, M., Panwar, S.S.), p.67, (Plenum Press 1994).
21. Dawes, N., Altoft, J., Pagurek, B., "Network Diagnosis by Reasoning in Uncertain Nested Evidence Spaces," *IEEE Transactions on Communications*, Vol. 43, p.466, 1995.
22. Cortes, C., Jackel, L.D., Chiang, W., "Limits on Learning Machine Accuracy Imposed by Data Quality," *Proceedings of NIPS94 - Neural Information Processing Systems: Natural and Synthetic Pagination,* p. 239, (MIT Press 1994).
23. Cox, R.M., "Detecting Lost Billing Records Using Kalman Filters," *AT&T Labs Preprint* (submitted), Oct. 1997.
24. Feather, F.E., Siewiorek, D., Maxion, R., "Fault Detection in an Ethernet Using Anomaly Signature Matching," *ACM SIGCOMM'93*, 23(4), 1993.
25. Maxion, R., Feather, F.E., "A Case Study of Ethernet Anomalies in a Distributed Computing Environment," *IEEE Transactions on Reliability*, 39(4), Oct 1990.
26. Hood, C., Ji, C., "Proactive Network Fault Detection," *IEEE Trans. Reliability*, Vol. 46, No. 3, p.333, 1997.
27. Hood, C., Ji, C., "Proactive Network Fault Detection," *Proceeding IEEE INFOCOM*, 1997.
28. Jakobson, G., Weissman, M.D., "Alarm Correlation," *IEEE Network*, p. 52, Nov 1993.
29. Katker, S., Paterok, M., "Fault Isolation and Event Correlation for Integrated Fault Management," *Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management*, p. 583, 1997.
30. Hasan, M.Z., Sugla, B., Viswanathan, R., "A Conceptual Framework for Network Management Event Correlation and Filtering System," *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management (IM'99)*, Edt. Edt. M. Sloman, S. Mazumdar, and E. Lupu, (IEEE Press), to appear in May 1999.
31. Stallings, W., "SNMP, SNMPv2, SNMPv3, and RMON 1 and 2," (Addison-Wesley, 1999).
32. Ho, L.L., Macey, C., Hiller, R., *in preparation*, 1999.