NOMAD:

Traffic-based Network Monitoring Framework for Anomaly Detection

Rajesh Talpade Gitae Kim
Telcordia Technologies (formerly Bellcore)
445 South Street
Morristown, NJ 07960
{rrt,kgt}@research.telcordia.com

Sumit Khurana
Computer Science Department
Colorado State University
Fort Collins, CO 80523
khurana@cs.colostate.edu

Abstract

Network performance monitoring is essential for managing a network efficiently and for ensuring reliable operation of the network. In this paper we introduce a scalable network monitoring framework, *(NOMAD)*, that detects network anomalies through the characterization of the dynamic statistical properties of network traffic. NOMAD relies on high resolution measurements and on-line analysis of network traffic to provide realtime alarms in the incipient phase of network anomalies. It incorporates a suite of anomaly identification algorithms based on path changes, flow shift, and packet delay variance, and relies extensively on IP packet header inforamtion, such as TTL, source/destination address and packet length, and router's timestamps. NOMAD can be deployed in a single backbone router or incrementally in a regional or large scale network for detecting and locating network anomalies by correlating spatial and temporal network state information.

Keywords: network performance monitoring; network anomalies; traffic analysis; time to live (TTL); flow; packet delay variance.

1 Introduction

Due to the phenomenal growth in network communication, networks with diverse networking technologies are being connected together at all scales of the network hierarchy ranging from the global Internet to corporate local intranets. Consequently, today's networks have become highly heterogeneous and vary greatly in the services they offer (both hardware and software) and the traffic that they carry. Aggregation of long-term and short-term sessions, bursty and moderately varying bit rates, real time and non-real time services using a highly dynamic protocol, have all given rise to traffic patterns and dynamics that are very particular to the expanding communications medium. While network heterogeneity provides more flexibility in utilizing the latest technologies and allows for customization by user applications, it also increases the risk of network faults or anomalies [21].

In addition to the heterogeneity of network traffic and services, the layering of network operations also makes network management difficult. Implementation modules such as the ISO/OSI layered reference model, where higher level protocols provide the procedure abstraction to shield the user from lower level implementation details, are commonly used. The procedure abstraction provides simpler network access when working with higher level protocols, but identifying problems occurring at the lower levels is often hard and makes the task of network monitoring and management more difficult [18].

One of the challenges of the next generation of networks is to provide a means for network management and control that is consistent with the underlying protocols and philosophy of growth. This need is especially true if one considers the growing use of intranets and the Internet for real-time applications such as corporate video conferencing and Internet Telephony. If the next generation of network technology is to operate beyond the levels of current networks, it will require a set of well-designed tools for its management. This does not imply a monolithic central mangement system but instead a dynamic and scalable network management system where anomalies can be reliably identified and corrected over a large and distributed network infrastructure.

Previous work in network monitoring include dependency graphs [14, 17], Management Information Base (MIB) variables [19], advanced databases [20], probabilistic inference [6], expert systems [1, 4, 11], and packet proving [2, 5]. A seminal work on net-

0-7695-0250-4/99 $10.00 © 1999 IEEE

work anomaly detection and identification is given in [15]. The problem with most of these schemes is that they heavily depend either on distributed interactions among monitoring stations within a network, or on additional control and probe messages. Both these approaches can easily lead to scalability problems. In addition, most of the above schemes are based on active methods that tend to use a high level of temporal and spatial resources to accomplish their goals.

In this paper we introduce NOMAD, a traffic-based network monitoring framework for anomaly detection. NOMAD is a scalable, passive network monitoring tool that adaptively characterizes the network state based on statistical inferences derived exclusively from network traffic stream measurements. Our focus in this paper is mainly on the functional and architectural concept of NOMAD and the suite of algorithms that it implements for the detection and identification of local anomalies.

The properties of dynamic network traffic can be categorized into three types: (i) path changes, (ii) changes in traffic flow, and (iii) variation in packet delays. Using these properties it is possible to define a model of network behavior that detects and identifies the characteristic signatures of network anomalies, such as router misconfiguration, overloading of router processes, and link overload. NOMAD relies on online, high resolution traffic measurements to provide real-time alarms in the incipient phase of network anomalies allowing early detection of network anomalies (*i.e.*, anomaly signatures), by implementing a suit of anomaly detection algorithms based on the three types of network properties. Our traffic measurements rely extensively on IP packet header information, such as TTL value, source/destination address, and packet delays. The incremental deployment of NOMAD can provide global diagnosis of the network state through the correlation of temporal and spatial information gathered throughout the network.

This paper is organized as follows. Section 2 gives a brief discussion of network anomalies. In Section 3, we introduce the concept of NOMAD and present details of anomaly detection and identification algorithms for NOMAD. Section 4 describes the network environment for data collection and presents our preliminary analysis on the traffic data. Finally, a summary and discussion on future work are given in Section 5.

2 Network anomalies

To clarify our problem domain, we discuss, in this section, the causal elements of anomalies in a general network environment. We also include a description of the various phenomena exhibited by an anomaly-burdened network.

The general categories of causal elements for network anomalies are as follows.

Router misconfiguration/malfunction: This includes manual misconfiguration of routers such as incorrect address allocation to a router port (resulting in false advertisements), false default route advertisement, routing loops and other network design errors, or a sudden router failure. It can be most readily detected by analyzing the traffic flows obtained from one or more monitoring points. Packet drops due to the expiration of TTL (Time To Live) values and faulty routing can occur under this situation.

Component failure and misbehavior: Down links and routers result in unplanned changes in network topology and re-routing of traffic. Excessive bit error rates (BER) in links, or power surges in routers/links can result in high packet loss. Changes in TTL value can occur under topology changes when traffic is re-routed.

Traffic overload: Over-loading of routers or links due to an increase in traffic. Packet drops (due to router buffer overflow) and delays (due to queuing delay in a router buffer) are most common signatures of traffic overload.

Network intrusion: Some examples include malicious attacks of the form of SYN attacks, email spam, and swamping of file servers. All can waste available resources by flooding with packets.

We will focus mainly on the first three categories of network anomalies, though some forms of network intrusion, such as swamping of file servers, can be detected through traffic stream analysis.

3 NOMAD: Concept

NOMAD is a passive network monitoring system that enables fast, accurate, and scaleable network management, capable of detecting, identifying, and locating network anomalies based on traffic stream measurements. It can be deployed in a local router to monitor activities of a partial network, or incrementally deployed in a wide-area or global environment to build a powerful monitoring tool that can locate and isolate anomalies by correlating global time and space information (Figure 1). The functional architecture of NOMAD consists of five functional modules (Figure 2).

Data Collector: This functional module is responsible for collecting real-time, high resolution traffic data in non-intrusive manner, and pass it to the Local Data Filter. The high resolution can help identify network anomalies accurately.

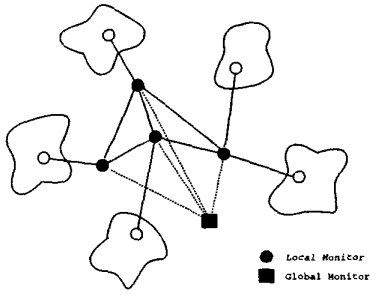

Figure 1: NOMAD - Local and Global Monitors

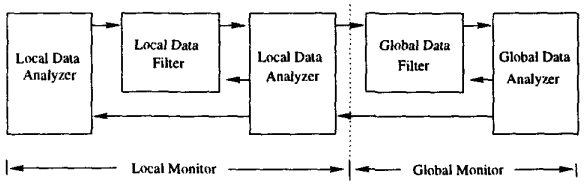

Figure 2: NOMAD – Functional Architecture

Local Data Filter: Its main function is to reduce the amount of data to be processed by the Local Data Analyzer, by filtering the traffic data. Source/destination IP address, subnet address, and type of application are some of the examples that can be effectively used in the data filtering process.

Local Data Analyzer: This module is responsible for detecting and identifying network anomalies through the analysis of traffic stream collected and filtered by the two preceding modules. Since it is the front-end module in NOMAD that actually analyzes collected data for detecting and identifying local anomalies, the overall performance of NOMAD depends significantly on the effectiveness of this module.

Global Data Filter: This module further reduces the data forwarded from the Local Data Analyzer. Again, source/destination IP address, subnet address, and type of application can be used as data filtering basis.

Global Data Analyzer: It correlates data from a number of local monitors to discover faults that would be undetectable locally. It can also locate faults in a more powerful way than the Local Data Analyzer, since it has a view of data that originates from many geographically distributed sources.

In the remainder of this paper we focus our discussion mainly on the Local Data Analyzer which is a critical functional component in the implementation of NOMAD. In particular, we elaborate on a suite of traffic analysis algorithms implemented in the Local Data Analyzer.

To detect and identify the characteristic signatures of network anomalies, such as router malfunction/misconfiguration, component failure and misbehavior, and traffic overload, we have classified the dynamic properties of network traffic that can be observed in networks layer into three categories: (i) changes in path and routing loops, (ii) abrupt changes in traffic flow, and (iii) variation in packet delays. These anomalous properties can be effectively detected through the careful observations of TTL values, traffic flows, and packet delays.

In the remainder of this section we describe a suite of anomaly detection and identification algorithms implemented in the Local Data Analyzer module. These algorithms are based on the extensive analysis of the TTL values, traffic flows and packet delays to monitor the network.

3.1 TTL-based analysis

When a link or router fails, the path from source to destination that goes through the failed component will be changed due to rerouting. One way of observing this situation is by looking into the TTL value carried in each packet from the same source (or the same subnet) and detecting the changes in the TTL value. In the following we introduce an anomaly signature detection algorithm based on TTL (Time to Live) values of observed data and a simulation model for validating the algorithm.

Detection of abnormal network behavior in order to detect network anomalies consists of two aspects: defining the *normal* and defining *deviations* from the normal, which signify anomalies. Also, since we are dealing with a large volume of data, any technique that is used should preferably not require storage of a large amount of data, while accurately depicting the state of the network. The notion of exponential averaging technique used for measurement of round trip time of IP traffic[13], and queue lengths in gateways[10], described in Appendix A of [13] can be applied for this requirement.

Starting with an estimator for the average m, the new average is defined as

$$m \leftarrow (1-w)m + w.T \qquad (1)$$

where T is the current measurement and w is a 'gain' which should be related to the variance of m . If w is chosen as a power of 2 the computation of m can be done using only integer arithmetic using shift operators.

A long term running average can be used to define normal behavior. A short term or *current* average should match the long term average if there is no abnormality. A significant difference in the current average from the long term average signifies deviant behavior.

3.1.1 Long term average

Averaging the value of TTL over a long interval of time for a particular network can help determine which values are in the acceptable range for that network.

The long term averaging can be carried out by using a small value of w, in order that the short term variations do not greatly alter the average. Only if the variation persists for a long time, will the long term average change significantly. This may indicate a permanent change in the route from that source.

3.1.2 Deviation from average

The state of the network can be said to have been altered if the values observed are consistently different from the long term average for a defined period of time, t. A naive approach would be to raise a flag every time a TTL value different from that observed over the past few minutes is observed. However the problem with this approach is that the jitter values vary a great deal. In time t very few packets may have been observed. We cannot accurately make a conclusion about the network from such little data. We can come up with better solutions using exponential averaging.

Let the current average be m. If for the next t time units we consistently see values of TTL equal to $m+D$, we want the average to reflect a change of X. If n packets are expected to arrive in time t, using equation 1 we get at the end of time t

$$(1-w)^n m + (m+D)w\sum_{i=1}^{n}(1-w)^{n-i} = m + X \quad (2)$$

$$\Rightarrow (1-w)^n m + (m+D)w\frac{1-(1-w)^n}{w} = m + X \quad (3)$$

$$\Rightarrow (1-w)^n = \frac{-D+X}{-D} \quad (4)$$

$$\Rightarrow \ln(1-w) = \frac{1}{n}\ln\left(1-\frac{X}{D}\right) \quad (5)$$

Solving for w,

$$w = 1 - e^{\frac{1}{n}\ln\left(1-\frac{X}{D}\right)}$$

By choosing the parameters n and $\frac{X}{D}$ appropriately a flag can be raised whenever this average varies from the long term average by a factor of $\frac{X}{D}$.

In the above expression we do not know the value of n. n is a function of the jitter, which varies over time. Moreover, internet traffic is bursty with relatively long inter arrival times between bursts. If few packets are seen in time t we wish to give more weight to each packet. A lone packet seen after a long interval is considered indicative of the state of the network during the entire interval, while packets seen as part of a burst are a manifestation of the same short term network state.

n can be recomputed as t/j based on the current estimate of jitter j. j can be computed by taking a short term running average for the jitter values. w can be recomputed with the new value of n.
On every packet arrival, the following computations need to be carried out

$$j \leftarrow (1-w_j)j + w_j c_j \quad (6)$$

$$n \leftarrow t/j \quad (7)$$

$$w \leftarrow 1 - e^{\frac{1}{n}\ln\left(1-\frac{X}{D}\right)} \quad (8)$$

$$m \leftarrow (1-w)m + wT \quad (9)$$

c_j is the current value of jitter and w_j, the gain for estimating the jitter value. T is the current TTL value. w_j should be set to a relatively high value (≈ 0.4) such that it reflects the current value rather than the long term average.

Keeping w constant take a sample every s time units, ignoring all intermediate values between s and $2s$. If no arrival occurs in this period repeat the previous value for the purpose of averaging. This method has the advantage of being easier to compute as it does not require recomputation of w. On the flip side if the value chosen is not representative of other values in the interval then the value of average is not accurate. The problem may be aggravated if a non representative value gets counted more than once due to no arrivals during the sampling period. This requires that the sampling period be carefully chosen; however the inter arrival time is not known *a priori* and can, moreover, change with time.
This method requires the following steps.

1. On every packet arrival
$$lval \leftarrow T$$

2. Every s time units
$$m \leftarrow (1-w)m + w.lval$$

3.1.3 Simulation

The above methods of average computation were tested using a simple simulation. Packets were assumed to arrive with an exponential inter arrival time at the rate of 5 packets per second with probability 0.8. With probability 0.2 packet arrival was at the rate of 0.5 packet per second. The TTL value was either 200 in a *good* state or 190 in a *bad* state. *Good* and *bad* states alternated with the duration of each being exponentially distributed. The mean length for a good state was 15 minutes whereas that for a bad state was 1 minute.

The simulation was run with a fixed value of w, and with varying values of w for different sampling intervals. The value of X/D was chosen to be 0.7, with $t = 60s$. The goal was to be able to detect all *bad* states that persist for more than $60s$. This would

be indicated by the average going below $193(200 + X/D * (190 - 200))$.

Using a fixed w, with a sampling interval of one packet every 5s, the average fell below 193 for several *bad* states lasting for less than 60s. Using a variable w with a $5s$ sampling interval, the average was significantly above 193 even for *bad* states longer than $60s$. When variable w was used and every packet considered for average computation the average fell below 193 *only* for those values for which the bad state lasted longer than 60s. Figure 3 shows the plots.

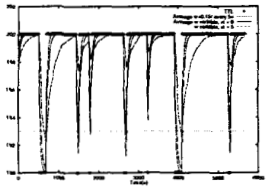

Figure 3: Simulation Output

Using a variable w along with sampling every packet provides a good metric for signaling *bad* states as an anomaly can be flagged every time the average crosses a certain threshold. This method was used on some actual data that we present in the next section.

3.2 Traffic flow analysis

The concept of a working set is used extensively for virtual memory and page management in computer systems [7, 8, 9]. Briefly, a set of highly accessed pages or memory blocks comprise the current working set for the system. This set can change over time as applications access different memory blocks. The challenge is to reduce the average latency for accessing a specific memory block by ensuring a high probability of finding the required memory block in the working set, rather than having to fetch and read it from memory. This can be achieved by using a large enough working set, or by other more sophisticated techniques such as block pre-fetching.

The working set concept can be applied to the NOMAD approach. Here the set would reflect high volume IP traffic flowing through the monitoring node. For each monitoring node, there would exist a set of flows (defined by source address, or source/destination pair) [12], such that the set remains relatively unchanged over a period of time. In the case of backbone nodes, this set should remain constant until the network topology is explicitly changed, or a routing problem or link/node outage results in a change in the traffic flow. In figure 4, for an example, node A's working set is ordinarily populated by subnets from networks X and Y. If link AC goes down, or a routing problem causes traffic from X and Y to flow over link DB, A's working set would change significantly.

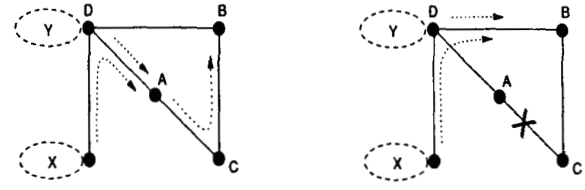

Figure 4: Shift in a Working Set

The working set can thus be used at a monitoring node to detect changes in traffic patterns, which may indicate a problem in the network. Parameters important for an effective working set mechanism are the entry/exit criteria and the working set size. A path can be inserted into the working set if satisfies the specified entry/exit criteria. A typical entry requirement can be, for e.g., all flows exceeding 100 packets per second. The entry requirements should be policy-driven, and should be able to capture the traffic pattern through the monitoring node. The exit requirement may be based on relative age of the entries, in that entries time-out and are deleted from the working set. In this case, the working set size is variable, and the node should be capable of occasionally accommodating a large working set. The working set size will be bounded above if the entry/exit criteria require an entry to be deleted before a new entry is inserted into a full working set. A monitoring node has to keep track of all the flows through it, and insert those flows satisfying the entry criteria into the working set. It also has to delete flows from the working set according to the exit criteria. An important requirement is a mechanism for the detection of a change in the working set. A simplistic approach is to maintain a snap-shot of the previous working set and compare it with the current set. The periodicity of the snap-shots determines the efficacy of the scheme. A high periodicity will result in better estimation of change, at the cost of increased processing at the monitoring node. Apart from detecting the change in traffic flows, a bi-level working set can be used for detecting overload conditions. The base level of the working set would be used as before for detecting traffic pattern changes. A second variable-size level would include flows that satisfy a specific criteria (e.g., very high throughput). The monitoring node can estimate the traffic load by the size of the second level working set, and can raise alarms if the size increases beyond a certain threshold.

3.3 Delay-based analysis

Packet delay variation is one of the most complex elements in network dynamics to characterize. At the same time, packet delay is potentially the richest source of information about the network since it reflects the queuing behavior of packets within the network.

There are three kinds of metrics for end-to-end packet delays that can be observed in either the network layer or transport layer: round trip time (RTT) and one-way transit time (OTT), and delay variances (*i.e.*, jitters). RTT measurement is an essential element for congestion control in transport protocols, but it may not serve as an accurate measurement of state in network path analysis since delays in forward and reverse are most often not symmetric [3]. OTT on the other hand can better serve the purpose of managing the network to detect and control anomalies by providing more accurate delay measurement in dynamic network environments [16, 3]. Jitters are becoming increasingly important as real-time applications become popular. Jitters can affect the performance of applicatios that require isochronous data streem. The Data Analyzer module relies on the delay variance measurement to detect and identify traffic overload.

One of the challenging issues in measuring packet delays has been clock accuracy since delay measurements rely on clock measurement. In general, for the accurate measurement of OTT, synchronization of the clocks between the source and destination is an important clock calibration requirement for packet delay measurements. In NOMAD the OTT measurement does not rely on the synchronized clock mechanism between the source and destination. Instead, it relies on the IP and TCP header information for a subset of TCP packets observed in the Local Data Collector.

We have designed a set of rather complicated rules to filter TCP packets in such a way that packets that are consecutively transmitted within a single TCP transmission window are identified and collected. We then infer the packet delays incurred from a source to a local monitor by measuring inter-packet arrival times among those consecutive packets. Note that the main purpose of measuring the packet delays is to estimate the rate of congestion along the path in the network. The delay values derived from this method may have the danger of overestimating the congestion rate when the number of hops between a source and a monitoring station becomes relatively large, since the observed delay reflects the accumulation of the transmission time and queuing delay at each link along the path. When the congestion is spread throughout in the path, the observed delay values that are accumulated along the path may not necessarily provide a concrete picture on the link congestion, since the delay values tend to become large as the source is separated from the monitoring station with a greater number of hops.

Another way of estimating the congestion is to induce an average congestion rate of the links in the path, by normalizing the observed delay values with the number of hops from a source to the monitoring station. We have also defined a set of rules to estimate the hop counts with a high level of accuracy, based on the TTL values in the IP headers. The normalized delay values can be used effectively to represent a scale-invariant congestion rate of the links in a path, regardless of the distance between a source and a monitoring station.

4 Traffic measurements

This section describes the network environment used for data collection, the data reduction method, and the result of our preliminary analysis on the traffic measurements for the three anomaly detection and identification algorithms that we introduced in the previous section.

Note that, since we implemented a local monitor at an edge of a network that lacks rigorous network dynamics caused by anomalies (*e.g.*, component failure and link congestion), the data set we collected does not comply fully with our original intention of verifying the effectiveness of the algorithms. However, we observe in the following that the algorithms capture the characteristic signatures that can occur in a limited network environment.

4.1 Network setup, data collection, and data reduction

A switch encountering all traffic from inside the Telcordia network to locations outside Telcordia was identified (Figure5). All packets that it encountered were mirrored onto one of it's output ports. A Sun sparc20 workstation was connected to this output port. Running tcpdump in promiscuous mode, the Sun workstation collected headers of all IP packets that it saw in the binary format supported by tcpdump.

Data was collected for 4 weekdays, Monday through Thursday for 3 weeks, 6/30/98, 7/2/98 to 7/16/98 and 8/04/98, starting at 8am each day. The dumps are organized into different sets, each containing headers for $3x10^6$ IP packets. A maximum of 96 bytes starting from the link layer header of an IP packet was stored for each packet, unless the headers including TCP and UDP headers, were shorter than

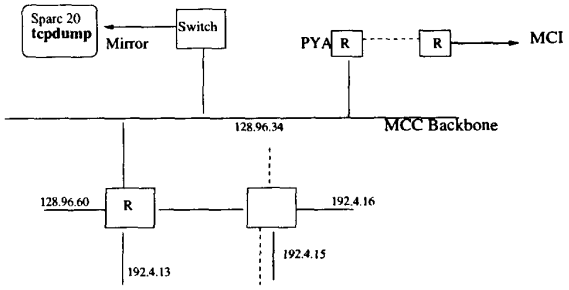

Figure 5: Data Collection Setup

96 bytes, in which case only the header bytes were captured.

Five such sets were generated everyday corresponding to $15x10^6$ packets. The typical time period that each file represented was 3 to 5 hours depending on the time of day with more frequent arrivals in the morning hours as compared to after 5pm.

Tcpdump shows the packet drops only when the process is killed during execution. Running tcpdump over two 24 hour periods when no data was collected showed that no packets were dropped. Since, no other processes were running on the machine when data was being collected, we expect that all packets encountered were captured on other days as well. The 5 sets were saved on tape daily for further processing.

4.2 Data analysis

4.2.1 TTL

A few data sets of $3x10^6$ packets were read from tape and analyzed as follows.

1. All incoming data into Telcordia was isolated.
2. This was counted by breaking up into bins based on the subnet IP address (first 3 bytes) of the source of the packets. All subnets with more than 1500 packets were identified.
3. The subnets identified in 2 were sorted in descending order based on the length of time over which the data packets were spread.
4. The top 25 subnets in 3 were designated to be the subnets of interest.
5. The TTL values and the jitter were plotted against time for each of these subnets.

The rationale behind following the above procedure was to isolate traffic that had been through a substantial number of hops from its origin for its characteristics to have been modified to reflect the state of the network, thus only incoming traffic was chosen. Considering flows over long period of times enables one to see the variations, while stipulating a minimum number of packets ensures that there is sufficient data to base one's conclusions on.

A preliminary study reveals that TTL values are fairly constant over the range of a few hours. Another interesting observation was that absolute values for TTL show wide variations, but if the difference from the nearest power of 2 is plotted, the plot continues to be a straight line. We speculate that the values are set to different powers of 2 (*e.g.*, 32, 64, and 256) by different hosts on the same subnet.

The jitter values showed wide variance as expected. This is directly related to the kind of service that is being accessed at the given instant of time. e.g. A telnet session may see minutes of delay when a user is not active, whereas a service like ntp may see periodic exchange of packets.

Average for Actual Data: Figures 6 shows plots for the average TTL of three contiguous data files. The method of average computation used is varying w ($t = 60s$), with every packet considered for computation of the average. The values were either 245 or 54, with values of 54 occurring less frequently. These could be said to correspond to the *bad* state. A drop below 111.3 would signify a value of 54 persisting for over $60s$. This was indeed found to be the case.

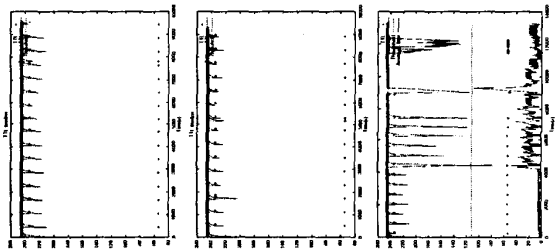

Figure 6: Data Set 1 (left), Data Set 2 (middle), and Data Set 3 (right).

Changes in TTL value: Here are some cases in which TTL values may show short term variations. Each of these cases needs to be studied in more detail to segregate normal behavior from what is a network anomaly.

1. Some applications / protocols like traceroute/ ntp assign special values to TTL field distinct from other IP packets originating from that host.
2. Some implementations such as Solaris 2.2 round robin between default hosts if more than one de-

fault host is specified. This may cause different IP packets to take different paths.

3. Routing loops due to a link or host going down. Routing tables may take time to stabilize in which time the packets may be routed incorrectly.

4. OSPF has different routes for different IP services.

Whereas, 1,2 and 4 correspond to normal behavior, 3 is an anomaly. Our method should be able to signal 3 as an anomaly while not raising a flag for the other cases. The value of the parameters X/D, s and w_j will be dependent on this.

4.2.2 Working set

We used data collected from our monitoring node to validate the working set theory described in the previous section. Before we state our observations, it is important to point out that the location of the monitoring node in the network influences the efficacy of the working set (WS) approach. A boundary node (such as ours) is not a good data collection point, since it captured all the traffic following into and out of Telcordia Applied Research (AR). The traffic flows into AR were primarily influenced by individual user applications, with changes in traffic patterns more indicative of the end of an application/session and the beginning of another. A monitoring node such as that indicated in Figure 7 in the previous section is able to capture the network traffic dynamics as the flows passing through it are not influenced by application usage.

Considering these facts, we analyzed the data from a two day period (07/15/98 and 07/16/98). Data flows were defined using source addresses only for incoming flows (from Internet to AR), since they represented data that had traveled over more hops, and hence better represented the network dynamics. Since the data exists in the form of 5 temporally contiguous data-sets per day, we retained this structure and isolated all incoming flows with more than 1500 packets from each data set for populating our working set. This approach corresponds to using a variable working set size, and an entry criteria of admitting flows with at least 1500 packets per 3 x 106 packets (size of each data-set). We compared the working set from each data-set with the succeeding set, which corresponds to using a very low periodicity (every 3 x 106 packets) for comparing the snap-shots (Table 1).

Column 2 in Table 1 indicates the working set size for a data set, while column 4 indicates the number of flows common to each working set pair over the two day period [1]. To capture the effect of working set size on the degree of commonality, Column 5 (*i.e.*, Percentage of Smaller WS Size) indicates the number of common flows as a percentage of the smaller working-set size in a pair, while column 6 (*i.e.*, Percentage of Total WS Size) indicates the same as a percentage of the total size of each working set pair under consideration.

From Table 1 we observe that, for a highly aggregated traffic (a window size of 3 to 5 hours), there exists little distinctive pattern in the numbers of common flows and no particular correlation between a pair of working set sizes and the number of common flows within the pair of working set. Since our goal here is to capture an area of flow transition where the number of common flows is abruptly changed as the window is advanced, we continued our experiments decreasing the window size. At the window size of 30 minutes, it started to show some identifiable patterns in the number of common flows and a rather consistent correlation between working set size and the number of common flows, and this pattern was clearly observable until the window size is reduced to 5 minutes. Figure 7 shows the working set size (left) and the number of common flows (right) under a window size of 5 minutes for the first 4 hours of the data set. It depicts four different levels of flow intensities, *i.e.*, each plot in the figure corresponds to the flow that generated at least 10, 25, 50, or 100 packets per 5-minute window. We employed a sliding window for processing the traffic data sets, which is advanced once in every 5 minutes. The working set in the previous window is compared to that in the current window to compute a set of common flows.

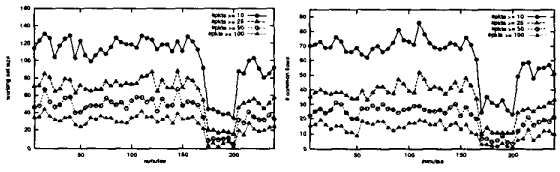

Figure 7: Working Set Size (left) and Number of Common Flows(right) are plotted on the y-axis as a function of time. The flows that generated at least 10, 25, 50, and 100 packets per 5-minute window were considered as working sets.

In Figure 7 we observe a consistency in the pattern between the working set size and the number of common flows. The sudden decrease and increase in both the working set size and the number of common

[1] we do not consider the last from day one and the first data-set from day two for such pair-wise analysis as they are significantly apart in time for any valid interpretation of the data.

Data Set Number	Working Set Size	Data Set Pairs	Number of Common Flows	Percentage of Smaller WS Size	Percentage of Total WS Size
0	89	0 - 1	18	58.06	15.00
1	31	1 - 2	16	51.61	22.86
2	39	2 - 3	17	43.59	18.68
3	52	3 - 4	18	34.62	16.22
4	59	4 - 5	43	55.84	26.71
5	84	5 - 6	34	48.57	23.13
6	77	7 - 8	28	48.28	21.88
7	70	8 - 9	21	84.00	25.30

Table 1: Working Set (WS) Analysis – Each working set constitutes all incoming flows with more than 1,500 packets from each traffic data set.

flows near 170 and 200-minute region may have between caused by user applications, network anomalies (*e.g.*, router failure), or both.

4.2.3 Packet delay variance

Using the data sets that we collected on 6/30/98, we measured packet delay variations for a subset of TCP data packets. As mentioned in the previous section, we identified TCP packets that were transmitted consecutively within a single window to capture the jitter value between the source and the local monitoring station. In addition, the hop count in the path that each packet travels was computed from the TTL value in the IP header, and used to normalize the jitter values.

Figure 8 shows the scatter plot of raw jitters (left) and normalized jitters (right) as a function of time for a subset of upstream TCP data packets, for the first 1,200 second period. The average of raw delay variable was measured at 217 msec while the average of normalized delay variable was 20 msec. The average hop count was 12.4 with a minimum and maximum hop count of 1 and 24 respectively.

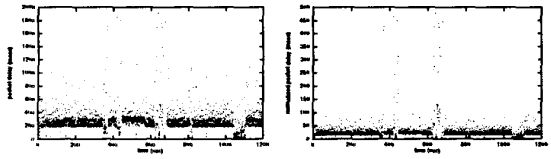

Figure 8: Raw jitters (left) and normalized jitters (right) are plotted on the y-axis as function of time.

For Figure 8, we implemented a sliding window with a window size of 10 seconds to compute the variation of variance and normalized delay variance. The standard deviation for delay variance increased over 4 and 7 times higher than other regions while the sliding window was pointing at 433 and 631 second respectively. This is a strong indication of network congestion at the regions with high standard deviation. The relatively high value of the normalized delay jitters at the two regions also suggests that it is highly probable that the congestion is occurring near the monitoring station.

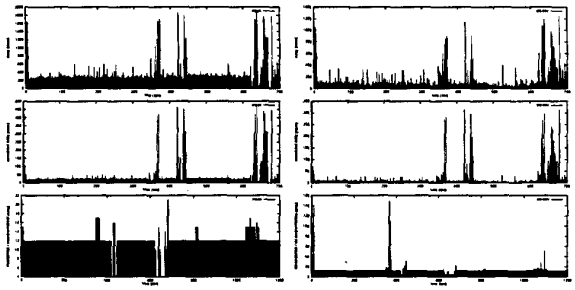

Figure 9: The mean of raw jitter (top left) and normalized jitter (middle left), and the standard deviation of raw work (top right) and normalized packet delay (middle right) are plotted on the y-axis as function of time. The ratio of raw jitters and normalized jitters for mean (bottom left) and standard deviation (bottom right) are plotted on the y-axis as function of time.

5 Summary

In this paper we introduced NOMAD — a scalable, passive monitoring framework that can characterize network anomalies based exclusively on statistical analysis of network stream measurements, and presented a set algorithms for anomaly detection and identification that can be implemented in local monitoring stations.

We have defined five top level functional modules for NOMAD: (i) Data Collector, (ii) Local Data Filter, (iii) Local Data Analyzer, (iv) Global Data Filter, and (v) Global Data Analyzer, to facilitate local and global network monitoring, and briefly described the main functions of each modules. The modular-based functional architecture of NOMAD provides an ability to deploy monitoring stations in incremental fashion in a wide-area or global environment to build a powerful monitoring tool that can detect and iden-

tify network anomalies through the correlation of spatial and temporal state information.

To detect the characteristic signatures of network anomalies (*e.g.*, router malfunction and misconfiguration, component failure and misbehavior, and traffic overload) we have classified the dynamic properties of network traffic into three categories, *i.e.*, (i) changes in path and routing loops, (ii) abrupt changes in traffic flow, and (iii) variation in packet delays, and devised three anomaly detection and identification algorithms based on the analysis of the TTL values, flow-based working set, and packet delays, that are incorporated in the Local Data Analyzer module in NOMAD. Our discussion in this paper was focused mainly on these local anomaly detection algorithms, since the performance of the global monitor in NOMAD depends much on the effectiveness of the local anomaly detection.

Finally, we have presented the result of our preliminary analysis of the traffic measurements to validate the anomaly detection and identification algorithms. Our result indicates that the experiments we carried on lack conclusive evidence of the effectiveness of the algorithms for detecting the network anomalies, mainly due to the following two reasons: (i) the traffic data sets lack generality, since they were collected at an edge of a network and (ii) it is not a trivial task to control real network to artificially induce anomalies to verify our methods. However, our traffic analysis shows that the anomaly detection algorithms we introduced behave consistently with our assumptions that can be inferred in the limited network setup.

References

[1] L. Bernstein and C. M. Yuhas. Expert systems in network management. *IEEE JSAC*, 6(5):784–787, June 1988.

[2] R. Carter and M. Crovella. Measuring bottleneck link speed in packet-switched metworks. Technical report, Computer Science Dept., Boston University, March 1996.

[3] K. Claffy, G. Polyzos, and H. Braun. Measurement considerations for assessing unidirectional latencies. *Internetworking: Research and Experience*, 4(3):121–132, September 1993.

[4] R. N. Cronk and P. H. Callahan. Rule-based expert systems for network management and operations: An introduction. *IEEE Network*, pages 7–21, September 1988.

[5] S. Dawson, F. Jahanian, and T. Mitton. Experiments on six commercial tcp implementations using a software fault injection tool. *Software: Practice and Experience*, 1997.

[6] R. Deng, A. Lazar, and W. Wang. A probabilistic approach to fault diagnosis in linear lightwave networks. *IEEE JSAC*, 11(9):1438–1448, 1993.

[7] P. J. Denning. Resource allocation in multiprocess computer systems. M.i.t project mac report mac-tr-50, M.I.T., Cambridge, Mass, May 1968.

[8] P. J. Denning. The working set model for program behavior. *Comm. of ACM*, 11(5):323–333, May 1968.

[9] P. J. Denning and S. C. Schwartz. Properties of the working set model. *Comm. of ACM*, 15(3):191–198, March 1972.

[10] S. Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, August 1993.

[11] J. J. Hannan. Network solutions employing expert systems. In *IEEE Annual International Phoenix Conference on Computer and Communications*, pages 543–547, 1987.

[12] C. Huitema. *IPv6: The New Internet Protocol.* Prentice Hall, Upper Saddle River, NJ, 1996.

[13] Van Jacobson. Congestion avoidance and control. In *Proceedings of SIGCOMM'88.* ACM, August 1988.

[14] Irene Katzela and Mischa Schwartz. Schemes for fault identification in communications networks. *IEEE/ACM Transactions on Networking*, June 1995.

[15] A. A. Lazar, W. Wang, and R. Deng. Models and algorithms for network fault detection and identification: A review. In *IEEE ICC*, Singapore, November 1992.

[16] Vern Paxson. *Measurements and Analysis of End-to-End Internet Dynamics.* PhD thesis, Computer Science, UC Berkeley, 1997.

[17] I. Rouvellou. *Graph Identification Techniques Applied to Network Management Problems.* PhD thesis, Columbia University, 1993.

[18] Steve Spanier. Design and implementation of a lan monitoring tool. *Computer Communications*, April 1988.

[19] Marina Thottan and Chuanyi Ji. Proactive anomaly detection using distributed intelligent agents. *IEEE Network*, September/October 1998.

[20] O. Wolfson, S. Sengupta, and Y. Yemini. Managing communication networks by monitoring databases. *IEEE Trans. on Software Engineering*, 17(9), 1991.

[21] Y. Yemini. A critical survey of network mangement protocol standards. In S. Aidarous and T. Plevyak, editors, *Telecommunciations Network Mangement Into the 21st Century.* IEEE Press, New York, NY, 1994.