Efficient Reactive Monitoring

Mark Dilman and Danny Raz, *Member, IEEE*

***Abstract*—Networks are monitored in order to ensure that the system operates within desirable parameters. The increasing complexity of networks and services provided by them increases this need for monitoring. Monitoring consists of *measuring* properties of the network, and of *inferring* an aggregate predicate from these measurements. Conducting such monitoring introduces traffic overhead that may reduce the overall effective throughput.**

This paper studies ways to minimize the monitoring communication overhead in IP networks. We develop and analyze several monitoring algorithms that achieve significant reduction in the management overhead while maintaining the functionality. The main idea is to combine global polling with local event driven reporting.

The amount of traffic saving depends on the statistical characterization of the monitored data. We indicate the specific statistical factors that affect the saving and show how to choose the right algorithm for the type of monitored data. In particular, our results show that for Internet traffic our algorithms can save more than 90% of the monitoring traffic.

***Index Terms*—Monitoring, network management.**

I. Introduction and Motivation

EFFICIENT management assumes having reliable information about the managed system. The only way to maintain such information at the management station is a continuous monitoring of the system parameters which affect management decisions. The increasing complexity of managed systems and services provided by them generates a need for monitoring of more and more parameters.

If the managed system is a network, the same links are often used to transfer both the payload and the monitoring data. In this case, the volume of the monitoring data being transferred directly impacts performance of the managed system. Therefore, minimizing the amount of monitoring related traffic in such networks is an important goal.

One can distinguish between two types of monitoring: *statistical monitoring* and *reactive monitoring*. In statistical monitoring, the management station derives some statistical properties, which are often used to predict some future trends, from the "raw" data. This basically means that all the "raw" data has to be transferred to the management station. In such a case, there is no big potential for reducing the monitoring traffic since all data must arrive at the management station.

Manuscript received March 8, 2000; revised December 20, 2000. This paper was presented in part by the IEEE Infocom 2001 Conference, Anchorage, AK, April 2001.

M. Dilman was with Bell-Labs Lucent Technologies, Holmdel, NJ 07733 USA. He is now with Oracle Systems, Redwood Shores, CA 94065 USA (e-mail: mark.dilman@oracle.com).

D. Raz was with Bell-Labs Lucent Technologies, Holmdel, NJ 07733 USA. He is now with the Department of Computer Science, Technion, Haifa 32000, Israel (e-mail: danny@cs.technion.ac.il).

Publisher Item Identifier S 0733-8716(02)03059-7.

Reactive monitoring is a case where the management station needs information about the network state in order to react (in real or semireal time) to certain alarm conditions that may develop in the network. Such conditions usually indicate either a fault or some anomalous behavior which may cause a fault later on. In this case, there is a good chance of finding a mechanism which minimizes the amount of data transferred to the management station. This paper concentrates on reactive monitoring.

Two basic techniques are used for network monitoring: polling and event reporting [11]. Polling is a process in which the management station sends requests to network elements in order to obtain the state information. Typically, polling is done periodically with the fixed frequency determined by the time window within which the alarm condition has to be detected. Event reporting is a process where a local event in a network element triggers a report, that is sent by that element to the management station.

In many practical network management applications, asynchronous traps can be defined on network elements so that event reporting can be used instead of explicit polling. This can be more efficient since an event is generated only when the value of a state variable of a network element reaches a certain threshold. However, in many cases there is a need to monitor a global system parameter which is defined as a function of local properties of different network elements. In order to monitor such global parameters using event reporting, local traps have to be emitted continuously with the fixed frequency which makes the event reporting as expensive as periodic polling.

Recently, a new theoretical framework for minimizing polling in the case of reactive monitoring was described in [6]. Their approach is based on the fact that the evolution of state variables is usually restricted by some constraints. Taking those constraints into account allows the management station to predict the future state based on the past information and perform polling aperiodically only when there is a possibility of an alarm condition. Their framework deals only with polling.

In this paper, we propose and analyze novel techniques which allow to significantly reduce the amount of monitoring related traffic. These techniques are based on a combination of aperiodic polling and asynchronous event reporting. Before we proceed to explain the main results of this paper, we briefly describe several scenarios where it can be used. These examples should help the reader understand the concept of reactive monitoring and the advantage of reducing the monitoring cost.

1) *Network Traffic.* A network management application is monitoring the overall amount of traffic from the organization subnetwork to the Internet. Once this amount exceeds some threshold some actions should be taken to ensure adequate service for the organization customers. Such actions may include: activating backup

0733-8716/02$17.00 © 2002 IEEE

lines, distributing more context from the organization web servers to their context delivery contractor, or restricting employees access to the Internet. Note that the organization may be connected to the Internet via several links, each located in a different site, and the function that is of interest is the sum of the local variables (interface out traffic).

2) *Mirror Load.* An organization Web site is distributed among several mirror sites. In order to optimize customer service and increase the sales, there is need to know which are the most popular pages. In other words, we would like to know (in real time in order to react) when the overall number of hits in the last five minutes, for a specific page exceeds some number. Note that again, we need to know when a function which is the sum of distributed values exceeds a threshold.
3) *Fighting Denial of Service Attack.* In order to fight a denial of service attack, the number of SYN packets arriving at the organization network is counted. Again, an action should be taken when the total number of such packets in the last minute is too large.
4) *Licensing Information.* In many cases, software licensing allows only a restricted number of users to use it simultaneity. If the software is installed in many machines, maintaining the actual number of active copies may become problematic. Note that we do not really need to know the actual number of users, but only to be alerted when this number exceeds the license threshold.
5) *Traffic Engineering.* In many proposed architectures, a central entity (Bandwidth Broker) is in charge of provisioning the quality-of-service (QoS) parameters of the routers in a subnetwork, and of negotiating with neighbor networks and/or incoming flows the possible level of available service. In order to do it in a cost effective way, the Bandwidth Brokers should receive feedback from the routers regarding the QoS parameters for the different flows. In many cases, the relevant information is just the sum of several variables from different routers (e.g., the total delay of a flow is the sum of the actual delay in each router on its path), and it is only important when this value is too big.

Note that the characterization of the data varies: the amount of different locations can vary from a few in the first two examples to several thousands in the last two, and the rate in which the data changes varies significantly among the different examples. However, in all the above examples there is a need to be alerted when the sum of several variables, each obtained in a different network location, exceeds a predefined threshold. Of course one can deploy a central algorithm that will poll all nodes periodically and will generate alarms as needed. The actual amount of communication needed for the monitoring process in such a case depends on the network we monitor and on the specific parameters we need to monitor. Consider again example 5). Assume that we monitor only five MIB variables using SNMP. The actual size of the packet containing the five values is about 1 kb. If we have 1000 routers in the network and we monitor each one every ten seconds, we get a rate of 100 kb/s only for the monitoring data. The problem is then how to achieve the same functionality with the least possible communication cost.

The main idea, as mentioned before, is to combine event reporting and polling. We derive two basic algorithms. The first based on a straight forward partitioning of the global resource to the separate nodes, giving a fixed budget to each of them. When the local variable exceeds this budget the node triggers a report, and the central manager issues a global poll. The second approach is rate based, a local element reports only when the rate in which the value changes locally, is too high. This allows the central manager to assume that as long as no report was received the change rates at each node is bounded. This is exactly the *integrity constraint* needed in the scheme of Jiao *et al.* [6]. In the latter case, we combine a centralized greedy algorithm with local rate traps. It turns out that performance of the different algorithms (i.e., the amount of messages needed to guarantee detection of all alarm conditions), depends heavily on the statistical characterization of the collected data, and the number of different nodes. We analyze the performance of the algorithms based upon this characterization and demonstrate the amount of saving one can get using them. We also present improved versions of the above algorithms and show that for real network traffic, (similar to the case described in the first example above), the amount of saving in monitoring traffic can be up to 97%. Note that while our main goal is to reduce the traffic overhead generated by the monitoring algorithm, our algorithms are also very efficient in terms of the processing time required by the centralized monitoring station. The dominating factor is the computation of the sum of the variables, which is linear in n the number of monitored variables. Moreover, in our scheme this computation is done only when required, and not at any polling interval. Thus, using our algorithms also saves in processing time at the management station.

In the next section, we formally define the model and the assumption we made. In Section III, we present the different algorithms and analyze their performance. In Section IV, we describe the methodology used, and in Section V present our simulation results. Based on these results, we derive improved algorithms and demonstrate their performance in Section VI. We end with a review of the related work in Section VII, and a short discussion.

II. Model and Assumptions

We are given n real-valued variables $x_1, \ldots, x_n$. For each x_i, we are given a fixed positive cost c_i, representing the cost of *measuring* x_i at any time. Time t is an integer, beginning at $t = 1$. Let x_{it} denote the value of x_i at time t. We are also given a global function $f(x_1, \ldots, x_n)$. The value of f at time t, $f_t = f(x_{1t}, \ldots, x_{nt})$, depends only on the values of the x_is at a single time, t. We also associate with f a global threshold value T. When the value of f exceeds this threshold an *alarm condition* occurs.

The alarm condition evaluation is done at node 0 by a centralized algorithm. The values of the different variables x_{it} is not necessarily known at this node. We distinguish between two different methods to get information related to the values of these variables:

1) Polling: the centralized algorithm is polling one or more variables. The decision to poll and the exact subset of the variables to be polled, is a result of a computation based on the information available to the centralized algorithm.

2) Event Reporting: a node initiates a report that may contain the value of the variable x_{it}. The report is triggered by some *local* event, which is a result of a local computation based on the values of $x_{it'}$ for $t' \leq t$.

We are interested in minimizing the communication cost required in order to detect alarm conditions. That is, we would like to minimize the measuring cost but still detect alarm conditions as soon as they hold. Note that we mainly consider communication complexity and the computational complexity of the algorithm's centralized component and we do not consider the complexity of computing the computation that trigger the local event. We also assume that communication is reliable and that the (communication) cost of polling variable x_i is the same as the (communication) cost of sending an event report for that variable.

The algorithm that decides which variables to measure, based upon values obtained in the past and the local event reporting, together with the algorithms that trigger the local event reporting, is called a *monitoring algorithm*. Such an algorithm is *correct* if it always detects alarm conditions, and is *optimal* if its cost is always no larger than the cost of any other correct algorithm. In general, optimal algorithms do not exist (see [6]), so in order to compare the effectiveness of the algorithm, we compare it to the algorithm that measures all the variables at all time steps (this is similar to an algorithm that generates an event report at each node at each time step) and then compute the value f and compare it to T. We call this obvious algorithm O_b. Clearly O_b is a correct algorithm but it has the largest cost.

Definition 1: The *cost ratio* of algorithm A with respect to a specific problem is r if there exists a constant c such that $Cost(A) \leq r \times Cost(O_b) + c$ for any instance of the problem, i.e., for any possible x_{it}s.

This is a worst case definition, it says that for long enough sequences the expected *cost per round* of algorithm A is no more than rn. In this paper, we concentrate on the case where $f = \sum w_i x_i$. This is both an important function by itself, and is general enough to capture much of the insight of the problem. Note that by using the log function, this case also covers functions like $f = \Pi x_i^{w_i}$. In the rest of the paper, we assume for simplicity that the costs c_i are identical for all nodes, the range of all local variables v_i is the same, and the weights w_i are one. We also assume a global time synchronization, and the algorithms are described in terms of a *step* which is one time unit. In practice, the time unit may be important, we discuss this issue further is Section V.

III. Basic Monitoring Algorithms

In this section, we present our basic monitoring algorithms. Both algorithms are based on combining polling and event reporting aiming at minimizing the overall communication cost. We separate the algorithms based on the way the event reporting is initiated. In the first algorithm a simple value threshold is used, and in the second a threshold on the value change rate is used. Recall that the alarm condition is $f = \sum_{i=1}^{n} x_i \geq T$.

A. Basic Algorithms

The first algorithm `Simple-value` is based on setting a local value threshold that triggers an event report. This threshold is set in a way that in order for the alarm condition to hold, at least one of the variables must exceed the threshold. In such a case, the centralized algorithm will poll all other variables and compare the function's value to the threshold.

For the formal description of the algorithm, we have to specify both the algorithm for the centralized monitoring process SV_C, and the algorithm for the distributed nodes SV_N. Algorithm SV_N is very simple: at each time t, if $x_{it} \geq T/n$ then send the value x_{it} to node 0.

The centralized algorithm SV_C is also simple: at each time t, if received one or more reports then poll all other nodes for their values. If $f = \sum_{i=1}^{n} x_i \geq T$ then generate an alarm.

It is not difficult to prove that the algorithm is correct. Note that the correctness of the algorithm does not depend on the values of the variables, or their statistical behavior.

Theorem 1: Algorithm `Simple-value` is a correct algorithm.

Proof: First note that if $f = \sum_{i=1}^{n} x_{it} \geq T$ then there is at least one i such that $x_{it} \geq T/n$. Hence, by the definition of SV_N, node i will send its value to node 0, and by the definition of SV_C, a polling will be done and the alarm condition will be detected. □

Next, we estimate the communication cost of this algorithm. Let $Pr(i)$ be the probability that the value of x_i is bigger than T/n. The algorithm checks all values at time t if and only if at least one value was above the local threshold T/n. This probability is one minus the probability that none of the values is over the local threshold. Therefore, the algorithm's expected cost per step is $n(1-\Pi_{i=1}^{n}(1-Pr(i)))$. If all variables are identical, and $Pr(i) = p$ for all is, then the algorithm's expected ratio is

$$\frac{n(1-(1-p)^n)}{n} = 1-(1-p)^n. \tag{1}$$

One clear disadvantage of this algorithm is that when n is big enough (with respect to p), the probability that a global poll is performed increases, and the cost ratio goes to one. On the other hand when the number of interfaces is small, the algorithm may perform well.

The second algorithm, `Simple-rate`, was designed to address this drawback. Instead of looking at the local value of the variable x_i, it looks at the *changes* in the value. The main idea is that if the central node assumes a bound on the value change per time unit, i.e., the first derivative of the value of each of the local variables, it can compute a safe bound for the time of the next necessary measurement. Assume that we know the current values $x_1, \ldots, x_n$, and we know that the value cannot increase by more than δ in each time unit, then we can compute the worst case (all values increase by the maximum amount) and delay any further polling until this time. Such an algorithm is called *greedy* in [6] where they proved that if it is guaranteed that the rate bound holds, then this is the best possible algorithm that uses only polling. In our case, no guarantee on the evaluation of the variables is given. Instead, we use an hypothesis like "the value change per step is bounded by δ" and if the hypothesis is proven to be wrong locally at a node, this node sends an event message to the central node. In such a case, the central node cannot guarantee that an alarm condition cannot occur, and a global polling is initiated. For the formal description of the algorithm, we have to specify again both the algorithm for

```
Simple-rate central
1. Init: t_m ← 0; f = 0
2. while (TRUE)
3.     if t ≥ t_m OR report RECEIVED
4.         f ← POLLALL(x_i)
5.         if f > T
6.             report ALARM
7.         else
8.             t_m ← t + ⌊(T − Σ_{i=1}^{n} x_it)/δn⌋
```

Fig. 1. The basic centralized rate algorithm.

the centralized monitoring process SR_C, and the algorithm for the distributed nodes SR_N. Algorithm SR_N is fairly simple: at each time t, if $x_{it} - x_{i(t-1)} > \delta$ then send the value x_{it} to node 0.

The centralized algorithm is a bit more complex (see Fig. 1): first initiate the variable t_m, that indicates the next time to poll, to be zero. Then at any time t, if $t \geq t_m$ or a report was received then poll all nonreporting nodes. Set t_m to be $t + \lfloor (T - \sum_{i=1}^{n} x_{it})/\delta n \rfloor$, and if $f > T$ report an alarm. Again it is not too difficult to see that Algorithm `Simple-rate` is correct for any variable distribution.

Theorem 2: Algorithm `Simple-rate` is a correct algorithm.

Proof: In order to show that `Simple-rate` is correct, it is sufficient to prove that if $x_{it'} - x_{i(t'-1)} \leq \delta$ for all time steps $t \leq t' \leq t_m = t + \lfloor (T - \sum_{i=1}^{n} x_{it})/\delta n \rfloor$, then $f(t') \leq T$ for all t'. This is true since the maximal amount of value change for every x_i is bounded by δ and the global change in f is bounded by $n\delta$. □

Next, we estimate the cost of `Simple-rate`. The cost is constructed from two components. The first one is due to the local events. Let $Pr_s(i)$ be the probability that the $x_{it} - x_{i(t-1)} > \delta$, then the probability that none of the variables had generated an event at time t is $\Pi_{i=1}^{n}(1 - Pr_s(i))$, so the probability that at least one event was generated is $1 - \Pi_{i=1}^{n}(1 - Pr_s(i))$, and the cost is n times this value.

The second part of the cost is due to the centralized polling part. We would like to compute the expected cost per step for this part. The problem is that the sampling process is very complicated and the next measuring time depends on the value of the current measurement. One way to estimate the value is to compute the expected number of steps that we can skip until the next measurement. We assume that the local rate change is bounded by δ, otherwise, an even driven polling will be generated. If all values were polled at time t, then the next time to poll (by the algorithm) is given by: $t + \lfloor (T - \sum_{i=1}^{n} x_{it})/\delta n$. Thus, ignoring a possible dependency among the points in time when polling was done, we can compute the expected number of steps per poll to be

$$EXP\left(\left\lfloor \frac{T - f(X)}{\delta n} \right\rfloor\right). \tag{2}$$

We can now estimate the overall expected cost per round of Algorithm `Simple-rate` by

$$n(1 - (1 - Pr_s(\delta))^n) + \frac{\delta n^2}{T - EXP[f(X)]} \tag{3}$$

where $Pr_s(\delta)$ is the probability that the rate change of x_i is larger than δ. Note that ignoring the floor function introduces an error factor, and (3) is just an estimation. Unlike the previous algorithm, in this case δ is a free variable and we can choose it to optimize performance. If we choose δ to be big, then $Pr_s(\delta)$ will be small and, therefore, the first term in (3) will be small. However, the second term in this equation will be big since it depends linearly on δ. From (3) we get that the optimal value for δ is the one satisfying the following equation:

$$(1 - Pr_s(\delta))^{n-1} \frac{dPr_s(\delta)}{d\delta} = \frac{-1}{T - EXP[f(X)]}. \tag{4}$$

The main factor here is the way $Pr_s(\delta)$ depends on δ. We will return to this point in Section V when we analyze the simulation results.

B. Processing Time

As mentioned before, our main goal is to reduce the traffic overhead generated by the monitoring algorithm. However, in many NM systems the bottle neck is the processing power of the centralized management station. Thus, it is important to verify that the new monitoring algorithms do not require an extensive amount of processing at the centralized station.

Consider the centralized part of Algorithm `Simple-value`. At each time step, if it receives a message from any element, it generates a general poll. Once all elements are polled the global value $f = \sum w_i x_i$ is computed. Since the obvious algorithm has to compute the global value at every time step, Algorithm `Simple-value` always uses less computation resources than the obvious one. In fact, since the computation of f is linear in the number of elements (n), the amount of saving in computation time is equal to the amount of saving in communication cost.

As for the centralized part of Algorithm `Simple-rate`, there is an additional computation of the value t_m, but this again can be done linearly in the number of elements and, therefore, again, the algorithm is much more efficient than the obvious polling algorithm.

IV. Methodology

We tested the performance of the algorithm under different statistical characterizations of the data, and for 2 to 12 interfaces. The algorithms were tested on three different types of generated data, and on real network traffic data collected from Bell-Labs Intranet via SNMP script. In all cases, we generated independent data for each of the (2 to 12) local variables. Note that if there exists a (positive) correlation between the different variables, it may improve the performance of all algorithms, since alarm conditions are more likely to be indicated locally.

Typically, as indicated by the examples in Section I, the variables we want to monitor evolve over time, and there is some type of time dependency. Thus, assuming independent variables at each step according to some distribution will not represent real data in an accurate way. Instead we model the time dependency in the following three types of data:

1) Uniform Change. In this type, the value of the variable x_i at time $t+1$ is given by: $x_{t+1i} = x_{ti} + y_{ti}$, where y_{ti} is a random variable distributed uniformly in a given range $[-k..+k]$. That is, the value of x_i at time $t+1$ depends only upon the value in the last time step, and the

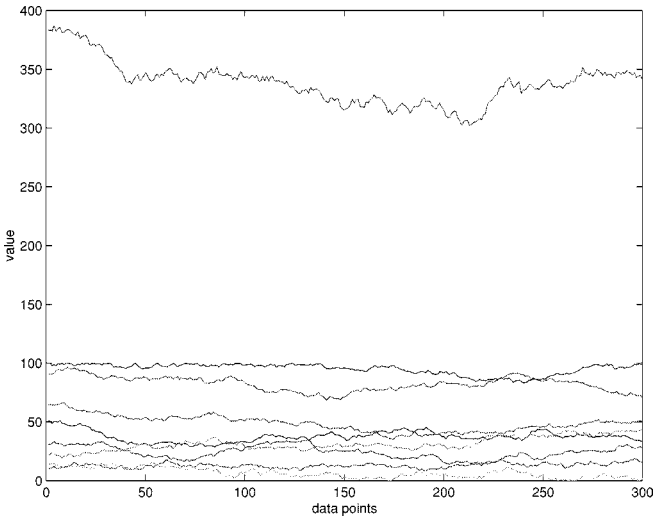

Fig. 2. Normal change traffic.

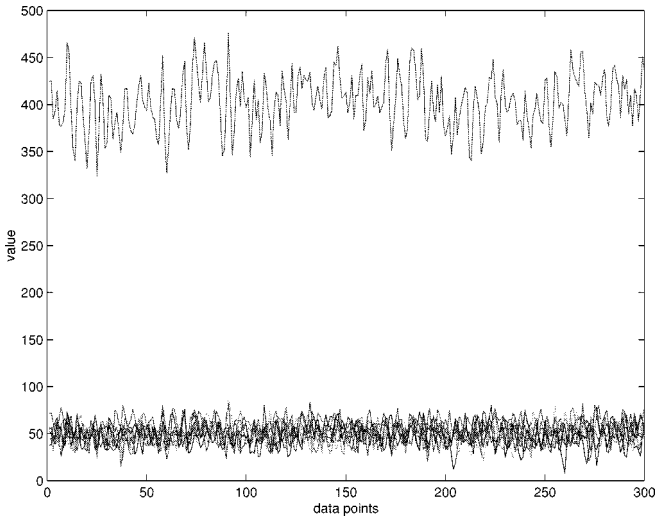

Fig. 3. Self-similar traffic.

change is uniformly over some given range $[-k..+k]$. For this simulation k was chosen to be 4% of the range of the variable x_i.

2) Normal Change. In this type, the value of the variable x_i at time $t+1$ is also given by $x_{t+1i} = x_{ti} + y_{ti}$, but the random variable y_{ti} is distributed in a normal distribution with mean value of zero and standard deviation k. Again, the value at time $t+1$ depends only upon the value in the last time step, but the change is distributed normally. k was chosen to be 1% of the range of the variable x_i. Fig. 2 shows the behavior of the variables x_i and the function $f = \sum x_i$. Note that the aggregation value is in some sense "smoother." Due to the random characterization of the increase/decrease factor, the variance of the sum of the variables is the sum of the variances. (See also [9] for a similar observation.)
3) Self-Similar. In order to capture stronger time dependency, we generated self-similar data, using the on/off mechanism described in [13]. Fig. 3 shows the behavior of the variables x_i and the function $f = \sum x_i$ in this case. As expected, the aggregated data look similar to the individual variables. A description of the self-similar characterization of aggregated self-similar traffic can be found in [9].

For all these three types of data, we chose a bounded range for the variables x_i. It was always positive and bounded by a predefined number k. Note, however, that the random process described to generate the data, may take the value of x_i outside of the specified range. When this happened we reversed the value of the random increase to decrease and vice versa. This created a higher probability for the value of the variables to be distributed closer to the bounds (0 and k). Another possibility, which we did not take, is to reset the value of the variable x_i to some random value in the range. This factor mainly affects the distribution of $T - f(X)$ but makes it easier to analyze $Pr_s(\delta)$. The target threshold was always 70% of the maximal possible aggregated value. That is, $T = 0.7nD$, where x_i is in the range $[0...D]$.

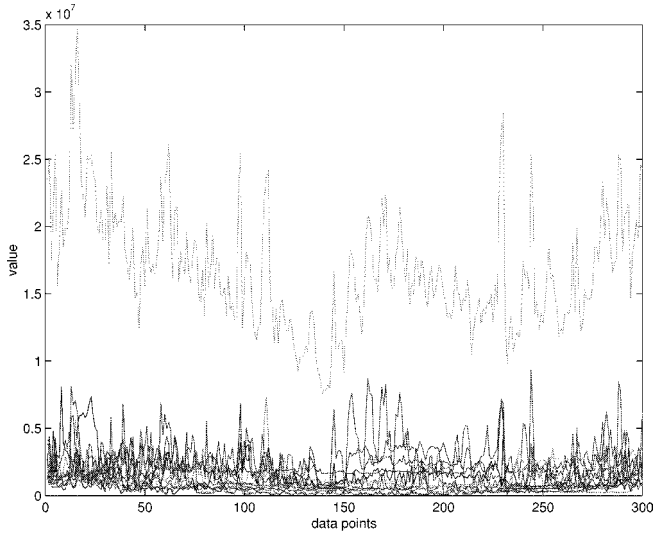

Fig. 4. Internet traffic, five minute average.

As mentioned above, the algorithms were tested also on real Internet traffic data. We collected the amount of outgoing octets from several WAN interfaces in the Bell-Labs network. The actual measurements were done every ten seconds, and were collected over the period of 48 hours. Out of this raw data, we computed the one minute average, and the five minutes average. In order to eliminate any possible correlation, we took separate segments of this data, starting at random times, and created a pool of measurements out of which we chose the data for each test. Fig. 4 shows the behavior of the variables x_i and the function $f = \sum x_i$ in that case. It looks like a combination of the two cases discussed before. We will come back to the important issue of the characterization of this data in Section V. For the setting of the global threshold, and the parameter δ in this case, we consider the maximal value that was reported in any interface in the interval as its maximal possible value (this is typically much smaller than the interface speed), and set up the global threshold to be the maximal aggregated value. This describes a situation where alarms do not occur often.

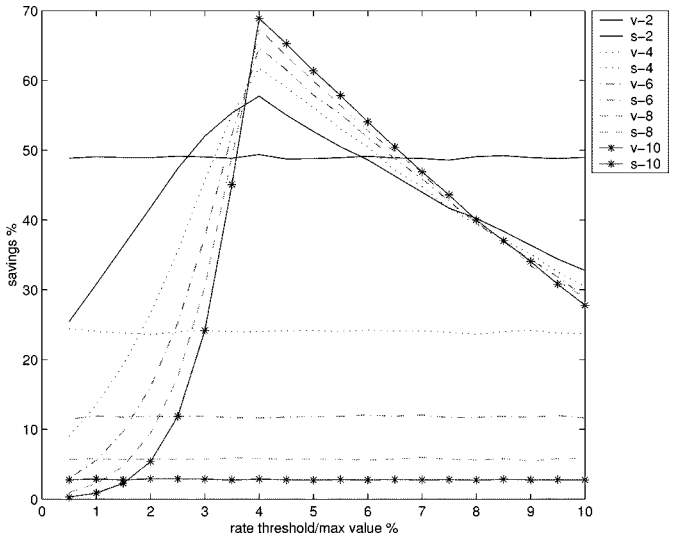

Fig. 5. Amount of traffic saving with uniform increment distribution data.

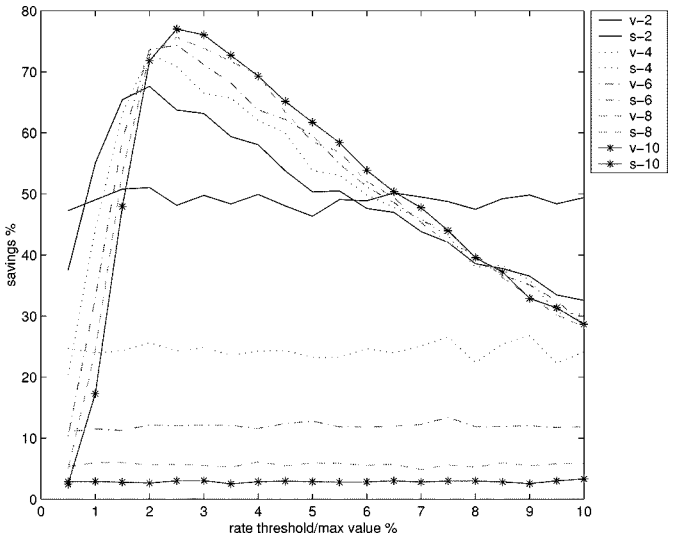

Fig. 6. Amount of traffic saving with normal increment distribution data.

V. Results

The amount of traffic saving achieved by running the basic algorithms on the uniform changed data is presented in Fig. 5. We plotted the amount of saving for different values of the rate threshold δ, given in percentage of the overall range of x_i. In the legend, we used v-l to indicate the performance of Algorithm `Simple-value` with l variables, and s-l to indicate the performance of Algorithm `Simple-rate` with l variables. Clearly, for Algorithm `Simple-value` the amount of saving does not depend on the value of δ. One can see that as expected (see Section III), the amount of saving achieved by `Simple-value` decreases as the number of variables increases. As for Algorithm `Simple-rate`, it is very clear from the figure that for any given n, there is a value of δ that achieves maximal saving. For this type of data, Algorithm `Simple-rate` performs better, and as n increases its relative performance improves dramatically. For 12 variables, for example, Algorithm `Simple-rate` saves 68% of the monitoring traffic, while Algorithm `Simple-value` saves only about 3%. An interesting phenomenon is that for δ values close to the optimal Algorithm `Simple-rate` saves more when the number of variables increases. This is not true when δ is smaller than 3.7% or larger than 8%. This behavior is not explained by (3) as a result of the estimation we did, by neglecting the affects of the floor function. Another interesting observation is that when n increases the optimal δ tends to a fixed value. This value (around 4%) is the maximal increase allowed in this experiment. The saving one could get from reducing δ below this value, by increasing the time between global polling, does not justify the extra cost due to event driven traps.

Fig. 6 presents the amount of saving achieved when the change in the value of the x_is was distributed normally. Again, it is clear from the graph that Algorithm `Simple-rate` performs better. This becomes more significant when n increases. The amount of saving achieved by Algorithm `Simple-value` is similar to the one in the uniform increment case. This is because this value depends only on the distribution of the values of the x_is, and this distribution is the same in the two scenarios. The optimal δ here is not achieved in a fixed value, but tends to increase when n increases. Recall that (3) has two terms. The first tends to zero very fast when δ exceeds twice the standard deviation, and thus for $\delta > 3$ the cost is almost linear in δ and the savings decreases accordingly.

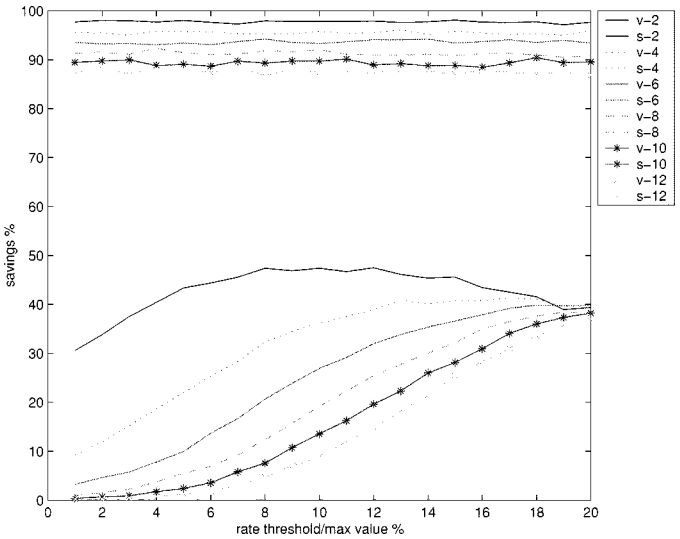

Fig. 7. Amount of traffic saving with self-similar data.

Fig. 7 presents the amount of saving achieved when the data is self-similar. The behavior is quite different here. One observation is that Algorithm `Simple-value` performs much better. This is due to the fact that the local values of each of the x_is are concentrated around their mean value (see Fig. 3). The second observation is that the dependency of the performance of Algorithm `Simple-rate` in δ does not have a clear maximal point in the relevant range. This is due to the fact that rate change distribution is heavy tailed, and thus $Pr_s(\delta)$ decreases fairly slow when δ increases.

The performance of the two algorithms on one minute and five minute average Internet traffic is presented in Figs. 8 and 9. The first observation is that for the threshold and number of interfaces tested (i.e., up to 12), `Simple-value` performs well,

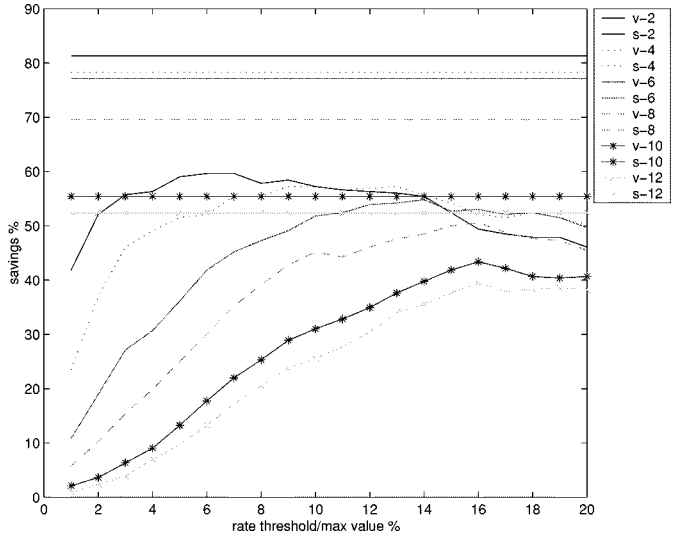

Fig. 8. Amount of traffic saving for one minute average traffic data.

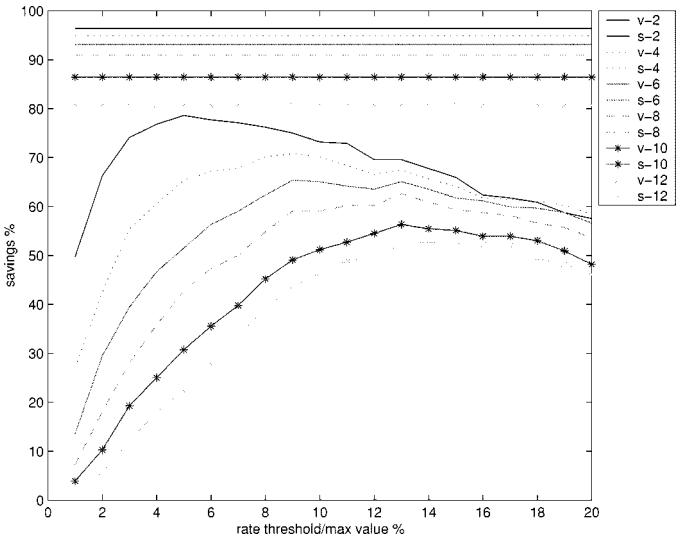

Fig. 9. Amount of traffic saving for five minute average traffic data.

and can save about 80% of the monitoring traffic for five minute average and 12 interfaces. The irregularity of the distances between the different lines, representing a different number of interfaces, can be explained by the (statistically) small number of different data points (332) used. As for Algorithm `Simple-rate`, it can save about half of the monitoring traffic (for the five minutes case), and its degradation with n is much smaller. An interesting fact is the difference between the five minute average and the one minute average. It looks as though the one minute data tends toward the self-similar type, while the five minute average graph has more of the characterizations of the normal increase data. The time scale of one to five minutes is interesting, since such data is commonly used in reporting Internet traffic and in many price schemes. It is evident from these results that this time frame is a transit stage from a self-similar characterization in smaller time frames (seconds) to a very well behaved distribution when considering a time frame of days.

Improved-rate local
1. $last\_update \leftarrow 0$
2. While TRUE
3. $\quad t \leftarrow CurrentTime$
4. $\quad$ if POLL
5. $\quad\quad last\_update \leftarrow t$
6. $\quad\quad v_{last\_update} \leftarrow x_{it}$
7. $\quad\quad$ send REPORT(x_{it})
8. $\quad$ else
9. $\quad\quad$ if $\delta > \frac{x_{it} - v_{last\_update}}{t - last\_update}$
10. $\quad\quad\quad$ send REPORT(x_{it})

Fig. 10. The improved rate algorithm for the local node.

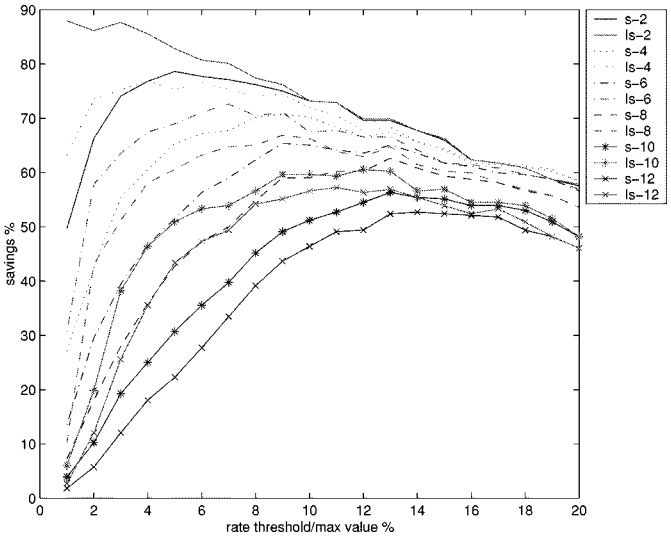

Fig. 11. Amount of traffic saving with Algorithm `Improved-rate` on five minute average data.

VI. Advanced Algorithms

Based on the performance of the basic algorithms, we present here improved versions and study their performance on the Internet traffic data.

We begin with an improved version of Algorithm `Simple-rate`. The main idea is that rather than check locally for the rate change in the last step, we compute the average change since the last global poll. If this value is below the rate threshold then it is safe, and no report should be sent. Fig. 10 presents this algorithm. The centralized part of it is the same as the version described in Section III.

It is not hard to prove that this algorithm is correct. Figs. 11 and 12 present the performance of algorithms `Simple-rate` and `Improved-rate` for the five minute and one minute average. As can be seen, the savings is about 10% and it decreases when the number of interfaces increases. This is because when the probability that at least one interface exceeds the threshold in the first rounds after the global poll is big, we cannot save much.

The main disadvantage of the value based approach is that when n becomes large, the probability of an event driven report (which then generates a global poll) increases. In order to reduce the cost in this case we want to reduce the "budget" given to each node, in a way that a single event driven report will not force the initiation of a global poll. If we have an upper bound on the value of x_i, say D, and we want to issue a global poll

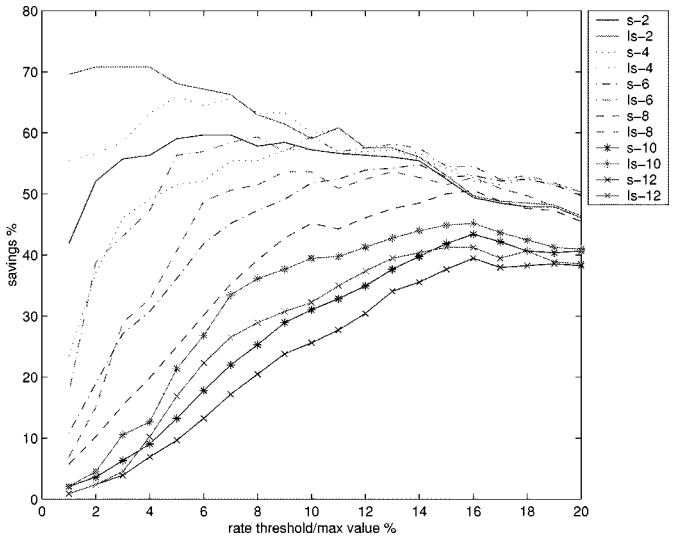

Fig. 12. Amount of traffic saving with Algorithm `Improved-rate` on one minute average data.

Improved-value central
1. while (TRUE)
2. $x_{ex} \leftarrow \sum RECEIVED(x_i)$
3. $n_{ex} \leftarrow$ number $RECEIVED(x_i)$
4. if $x_{ex} + (n - n_{ex})T_l > T$
5. $f \leftarrow POLLALL(x_i)$
6. if $f > T$
7. report ALARM

Fig. 13. The improved centralized value algorithm.

only when l local variables exceed the local threshold then in order to ensure the correctness of the algorithm we should have

$$(l-1)D + (n-(l-1))T_l < T$$

or

$$T_l = \frac{T-(l-1)D}{n-(l-1)}$$

where T_l is the local threshold. The problem, of course, is that the probability for local events increases as the local threshold decreases. The second problem is that in real interface data we do not have a strict bound D. The interface speed, which can be used for this, is in most cases an overestimation, and using it will degradate the algorithm's performance. To overcome this problem we slightly modified the algorithm and used x_i real value in the event report message. Thus, the node part of Algorithm `Improved-value` is very simple: whenever $x_{it} > T_l$ send the value of x_i. The centralized part of Algorithm `Improved-value` is presented in Fig. 13. In each round it collects the event driven messages, computes the sum of the received variables, and if there is a possibility of exceeding the global threshold it issues a global poll.

Unlike Algorithm `Simple-value`, here we can choose the parameter T_l. Figs. 14 and 15 present the performance of Algorithm `Improved-value` on the five minute and one minute average data.

The x axis in the figures shows the ratio between the chosen value of T_l, and the threshold for the basic algorithm which is T/n. Clearly, T_l cannot be larger than T/n because in such a case we cannot guarantee the correctness of the algorithm.

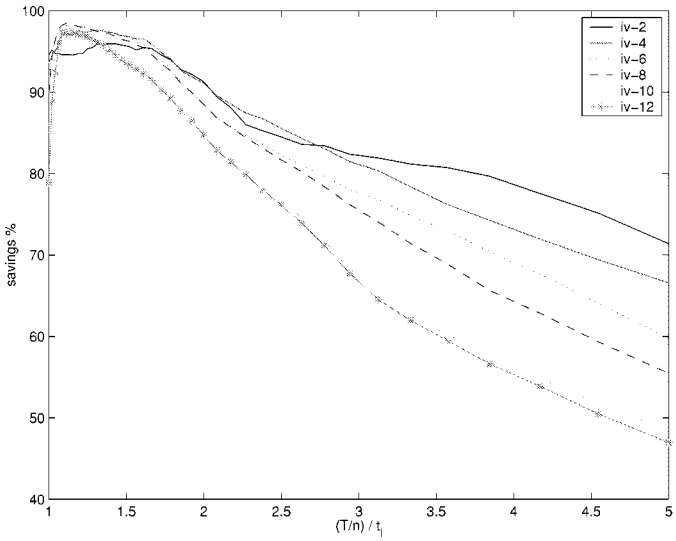

Fig. 14. Amount of traffic saving with Algorithm `Improved-value` on five minute average data.

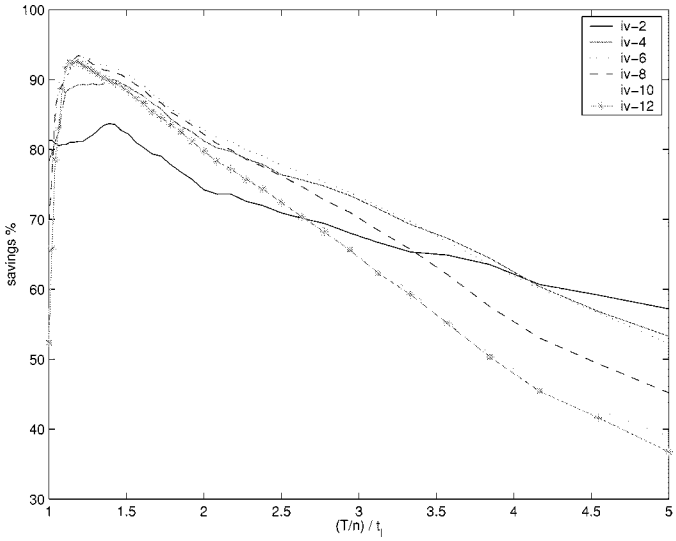

Fig. 15. Amount of traffic saving with Algorithm `Improved-value` on one minute average data.

One clear observation is that using the additional freedom of choosing T_l, reduces dramatically the dependency in n, and by tuning the algorithm according to the number of interfaces, we achieve a similar amount of savings for all tested ns. The amount of savings, 97% for the five minute average data, and 92% for the one minute data is significant, and indicates that this algorithm may perform well for different time scales.

VII. Related Work

The question of how to monitor integrated networks was addressed by the work of Mazumdar and Lazar [8]. They mainly considered the problem of how to decide which variables should be monitored, and how to specify the appropriate ranges, so that the information required to achieve a certain management goal is available. More recent works deal with the problem of achieving high level management goals while maintaining the amount of system resources used for management purpose as small as possible. In [4], the polling frequency is changed based

either on a threshold or history measurements. In [7], polling frequency is adaptively tailored to a specific application. In [14], the available bandwidth for the monitoring was taken into account when adjusting the polling rate. However, while these methods may reduce the amount of resources used for polling in certain scenarios, they do not deal with the functionality needed from the monitoring algorithm.

Several papers addressed the question of proactive anomaly detection. In [12], Thottan and Ji studied a similar problem of trying to identify in real time, signs that indicate an upcoming failure. Their work concentrated on the ways one can predict such failures and on the actual variables that needed to be monitored. In [5], Ho *et al.*described the design and implementation of a system that detects anomalies in pseudo real time. Again, the focus in this work is the actual variables that have to be monitored and the (on-line) computation of the different functions and thresholds that indicate the alarm condition.

The first work that proposed a model to study the communication overhead of reactive monitoring was [6].[1] However, Jiao *et al*. considered only polling, and did not consider event driven reports in their model. Furthermore, their work assumes a certain knowledge of integrity constraints, that restrict the evolution of the variables. In practice such constraints are either hard to discover, or do not exist, hence, the applicability of the model was not clear. In this paper, we introduce the idea of combining local event driven reports with the global polling. This allows us to use some of the results from [6] in a practical setting.

The problem of efficient reactive monitoring relates to several problems in distributed data bases. One can look at it as trying to verify that the relation $\sum x_i < T$ holds. Such inequality constraints are considered for example in [2]. However, the focus of the work in this area is to guarantee conditions of the data base, while we have no control over the variables we monitor.

The statistical characterization of Internet traffic has been the subject of much research interest in the past few years, see for example [3], [10], and [13]. More specifically, recently a similar topic of observing a mixture of self-similar and Poisson behavior was studied in [1] and [9]. However, we approach the problem from a different angel since our interest in this problem comes from the performance analysis point of view, where we want to find the best algorithm suitable for the data we are monitoring.

VIII. Discussion and Future Work

The main contribution of this paper is the design of practical efficient monitoring algorithms. We have demonstrated how the combination of a central monitoring algorithm, with simple local constraint verification, that fits naturally into the SNMP framework, can be used to save a significant amount of the monitoring overhead. Moreover, these algorithms also reduce the amount of computation needed at the management station. It turns out that the tailoring of optimal performing algorithms depends on the data characterization. We explored this point and identified the function $Pr_s(\delta)$, as playing an important role in the amount of saving achieved by our algorithms.

Another important issue is the affect of the actual threshold value on the performance of reactive monitoring algorithms. The value of the threshold is determined by the application, i.e., it comes from the definition of the alarm condition. However, the performance of the algorithms depend on the distance of the actual value from the defined threshold, and on the actual distribution of the values. A more rigorous study of this interesting relationship is left for future study.

[1]Although they did not use the name reactive monitoring.

References

[1] R. G. Addie, T. D. Neame, and M. Zukerman, "Modeling superposition of many sources generating self similar traffic," in *Proc. ICC '99*, Vancouver, Canada, June 1999, pp. 387–391.

[2] D. Barbara and H. Garcia-Molina, "The Demarcation protocol: A technique for maintaining linear arithmetic constraints in distributed database systems," in *Proc. Int. Conf. Extending Database Technology*, Vienna, Austria, Mar. 1992, pp. 373–388.

[3] M. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: Evidence and possible causes," presented at the SIGMETRICS '96, 1996.

[4] P. Dini, G. V. Bochmann, T. Koch, and B. Kramer, "Agent based management of distributed systems with variable polling frequency policies," presented at the Integrated Network Management Symp. (IM 97), San Diego, CA, May 1997.

[5] I. L. Ho, D. Cavuto, S. Papavassiliou, and A. Zawadzki, "Adaptive and automated detection of service anomalies in transaction-oriented wan's: Network analysis, algorithmic, implementation, and deployment," *IEEE J. Select. Areas Commun.*, vol. 18, no. 5, pp. 744–757, May 2000.

[6] J. Jiao, S. Naqvi, D. Raz, and B. Sugla, "Toward efficient monitoring," *IEEE J. Select. Areas Commun.*, vol. 18, pp. 723–732, May 2000.

[7] P. Moghe and M. Evangelista, "An adaptive polling algorithma," presented at the IEEE/IFIT Network Operations and Management Symposium (NOMS 98), New Orleans, Feb. 1998.

[8] S. Mazumdar and A. A. Lazar, "Objective-driven monitoring for broadband networks," *IEEE Trans. Knowledge and Data Eng.*, vol. 8, June 1996.

[9] R. Morris and D. Lin, "Variance of aggregated web traffic," in *IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 2000, pp. 360–366.

[10] V. Paxson and S. Floyd, "Wide-area traffic: The failure of poisson modeling," in *Proc. IEEE/ACM Transactions Networking*, vol. 3, June 1995, pp. 226–244.

[11] W. Stallings, *SNMP, SNMPv2, SNMPv3, RMON1 and 2*. Reading, MA: Addison-Wesley, 1998.

[12] M. Thottan and C. Ji, "Proactive anomaly detection using distributed intelligent agents," *IEEE/ACM Network*, pp. 21–27, Sept.–Oct. 1998.

[13] W. Willinger, M. Taqqu, R. Shermann, and D. V. Wilson, "Self-similarity through high-variability: Statistical analysis of ethernet lan traffic at the source level," *IEEE/ACM Trans. Networking*, vol. 5, pp. 71–86, 1997.

[14] K. Yoshihara, K. Sugiyama, H. Horiuchi, and S. Obana, "Dynamic polling scheme based on time variation of network management information values," in *Proc. Integrated Network Management Symp. (IM '99)*, Boston, May 1999, pp. 141–154.

Mark Dilman received the Ph.D. degree in computer science from the Russian Academy of Sciences, Moscow, Russia, in 1993.

From 1993 to 1994, he worked as a consultant for AT&T Bell Labs, Holmdel, NJ, developing systems for distributed simulation and software verification. From 1994 to 1997, he was with Computer Associates where he implemented new features for the Ingres database server that improved concurrency control and data consistency. During 1997–2000, he was a Member of Technical Staff at Lucent Bell Labs, Holmdel, NJ, and worked on various projects in network fault, performance and security management. Currently, he is a Principal Member of Technical Staff at Oracle Corporation, Redwood Shores, CA, where he is responsible for improving availability and fault tolerance of the Oracle database server.

Danny Raz (M'97) received the doctoral degree from the Weizmann Institute of Science, Rehovot, Israel, in 1995.

From 1995 to 1997, he was a post-doctoral fellow at the International Computer Science Institute, (ICSI) Berkeley, CA, and a Visiting Lecturer at the University of California, Berkeley. Between 1997 and 2001, he was a Member of Technical Staff at the Networking Research Laboratory at Bell Labs, Lucent Technologies, Holmdel, NJ. On October 2000, he joined the faculty of the Computer Science Department at the Technion, Israel. His primary research interest is the theory and application of management related problems in IP networks.

Dr. Raz served as the General Chair of OpenArch 2000, a TPC member for INFOCOM 2002, OpenArch 2000–2001 IM 2001, NOMS 2002, and GI 2002, and an Editor in the Journal of Communications and Networks (JCN).