Robust Resource Allocation for Online Network Monitoring

Pere Barlet-Ros[#], Josep Sanjuàs-Cuxart[#], Josep Solé-Pareta[#], Gianluca Iannaccone[*]

[#]*Computer Architecture Dept., Technical University of Catalonia (UPC), Spain*
{pbarlet, jsanjuas, pareta}@ac.upc.edu

[*]*Intel Research Berkeley, United States*
gianluca.iannaccone@intel.com

***Abstract*— Building robust network monitoring applications is hard given the unpredictable nature of network traffic and continuous growth of link speeds, data rates and complexity of traffic analysis tasks. Effective resource management techniques are now a basic requirement for this class of applications, which have to deal inevitably with the effects of extreme overload situations during their normal operation. In this paper, we present in detail the problems involved in the management of system resources in network monitoring and describe the design of a load shedding scheme that can efficiently handle extreme overload situations by gracefully degrading the accuracy of monitoring applications. Our method controls the resources allocated to each application by dynamically adjusting the sampling rate based on an online prediction model of the system resource requirements. We present experimental evidence of the robustness and performance of our system using real traffic traces and injecting synthetic traffic anomalies.**

I. INTRODUCTION

Passive network monitoring systems are commonly used to collect the traffic from operational networks and compute a set of relevant metrics in real time. The information they provide is crucial for managing and operating data networks, and it is used to aid network operators in the tasks of traffic engineering, capacity planning, network troubleshooting or anomaly detection, among others.

The information that network operators desire to extract from the network traffic is of different size, granularity and accuracy depending on the measurement task. For example, the relevant data for capacity planning and intrusion detection are very different. To satisfy these different demands, a new class of network monitoring systems is emerging to handle multiple arbitrary network traffic queries. These systems allow users to easily define monitoring applications by providing them with an abstraction layer of the underlying network technology and hardware details.

The main challenge in network monitoring is to achieve robustness against overload situations due to the continuous growth of link speeds, data rates and complexity of traffic analysis methods. This problem is even harder considering that these systems have to handle a large number of competing traffic queries in a resource constrained environment. Unfortunately, over-provisioning the system for worst case scenarios is prohibitively expensive, because it is unviable to dimension system buffers to absorb sustained peaks in the case of anomalies or extreme traffic mixes. However, it is right under these situations when network operators most value the results of their queries. Therefore, it is essential to prevent uncontrolled packet losses during overload situations to minimize their impact on the accuracy of the results.

In this paper, we introduce the problem of resource management in the context of network monitoring and discuss why this is an interesting and challenging problem (Section II). We review the design of an online predictive resource management system, firstly presented in [1], which allows network monitoring systems to efficiently handle extreme overload situations (Sections III-V). Finally, we evaluate the robustness and performance of our solution in the presence of overload situations due to anomalous traffic patterns, and show their impact on the accuracy of the results (Section VI).

II. BACKGROUND

In a distributed network monitoring infrastructure, there are two possible resource management actions to address overload situations. The first consists of trying to solve the problem locally (e.g. to apply sampling where an overload situation is detected). The second option is to take a global action (e.g. to distribute excess load among the monitors of the infrastructure). If no actions are taken in a timely manner, queues will form increasing response delays and, eventually, the platform will experience uncontrolled packet losses, leading to a severe impact on the accuracy of the results.

Local resource management techniques are needed to manage the available system resources in a single monitor, according to a given policy. For example, such a policy might reduce query response times, while minimizing the impact of overload situations on the accuracy of the results. Simply rejecting incoming queries is not an option, since queries already running in the system may also exceed the system capacity. We refer to this problem as the *local resource management problem*.

Given that new network monitoring platforms are distributed systems in nature, global decisions to overcome overload situations can also be taken. Global resource management techniques are used to distribute the queries among the multiple monitoring systems in order to balance the load of the infrastructure. However, traditional load-balancing and load-sharing approaches used in other contexts are usually

This work was funded by a University Research Grant awarded by the Intel Research Council, and by the Spanish Ministry of Education under contracts TSI2005-07520-C03-02 (CEPOS) and TEC2005-08051-C03-01 (CATARO).

	Offline	*Online*
Local	• Static assignment of queries • Resource provisioning ○ CPU, memory, etc.	• **Query scheduling** • **Load shedding**
Global	• Placement of monitors • Placement of queries	• Dissemination of queries • Load distribution

Fig. 1 Resource management problem space

not suitable for network monitoring. The main reason is that neither the incoming traffic nor most queries can be easily migrated to other monitors, since the interesting traffic resides on the network where the monitor is attached to. We refer to this problem as the *global resource management problem*.

On the other hand, some resource management decisions can be taken statically (i.e. at configuration time) or dynamically (i.e. at run time). Therefore, we can divide the resource management problem space in the context of network monitoring systems into four dimensions (see Fig. 1), according to whether decisions are taken offline or online, and if they are local (i.e. in a single monitor) or global (i.e. involving multiple monitors):

1. The *local static resource management problem* can be divided into two different sub-problems: *(i)* provisioning of system resources (i.e. CPU, memory, I/O bandwidth, storage space, etc.) according to network properties (e.g. network bandwidth, traffic characteristics, etc.) and *(ii)* static planning of a fixed set of queries to be executed in the network monitor.
2. The *global static resource management problem* refers to the placement of both monitors over the network (i.e. where to place the network monitors according to a given budget and/or measurement goals [2]) and the static distribution of queries over the available monitors.
3. The *local dynamic resource management problem* consists of managing the local queries given the available resources to ensure fairness of service and maximize the utility of the system according to a given policy.
4. The *global dynamic resource management problem* basically refers to how to distribute the load of the platform among the multiple monitors in an effective and efficient manner.

In the rest of this paper we focus on the local dynamic resource management problem, although the feasibility of using some of the proposals resulting from this work to address the global resource management problem constitutes an important part of our future work.

A. Traditional Systems vs. Network Monitoring Systems

Resource management techniques have been extensively studied in other contexts, such as operating systems, distributed systems or database management systems. However, network monitoring systems have several particularities that render solutions adopted in other contexts unsuitable. These differences can be summarized as follows:

1. *Arbitrary input*. Traditional resource management techniques have been designed for pull-based systems, where data feed rates can be easily managed, given that the relevant data reside on disk. On the contrary, in network monitoring the input data is the network traffic, which is generated by external sources that cannot be controlled by the monitoring system. Network traffic is highly variable and unpredictable in nature, and typically peak rates are several orders of magnitude greater than the average traffic. Thus, provisioning a network monitoring system to handle peak rates is not possible. However, it is usually during these bursts when the monitoring system is mostly needed and results may be more critical (e.g. to detect network attacks or anomalies). For this reason, network operators are particularly interested in capturing the properties of the traffic during overload situations.
2. *Data rates and volume*. The input rates and volume of data in an online network monitoring system are usually extremely high (e.g. 10 Gbps). Traditional pull-based systems do not target the high data rates involved in network monitoring. This makes traditional approaches, where data are firstly loaded into static databases, unviable in this scenario.
3. *Arbitrary computation*. On the one hand, query loads heavily depend on the incoming traffic, which is unpredictable in nature. On the other hand, query loads depend on its actual implementation, which is also arbitrary. In particular, new network monitoring systems allow users to express queries with arbitrary resource requirements (e.g. written in imperative programming languages). As a result, most queries do not have a fixed cost per packet. For example, a worm detection query may be idle for a long period of time until attack traffic appears in the network.
4. *Real-time results*. Most pull-based resource management techniques assume that applications do not have real-time requirements. On the contrary, most network queries require a timely response, whereas some of them may even come with an explicit deadline the monitoring system must assure. For those queries, late results may be useless (e.g. virus and worm detection queries).

B. Related Work

Recently, several research proposals have addressed these challenges in the context of network monitoring and data stream management systems.

In the network monitoring space, existing solutions consider only a pre-defined set of well-known traffic metrics. To deal with overload situations, the accuracy of this fixed set of metrics is degraded by means of traffic filtering, dynamic sampling and/or aggregation techniques. Examples of monitoring systems implementing these methods include Adaptive NetFlow [3] and the robust traffic summaries proposed in [4].

Data stream management systems are push-based systems specifically designed for continuous query processing. One of the major challenges in these systems, like in network monitoring, is to support real-time processing with limited system resources. Thus, research proposals in this area are very relevant to our work. They include solutions that define a declarative query language, with a limited set of operators, for

which the cost and selectivity is assumed to be known. In the presence of overload, operator-specific load shedding techniques are implemented, such as selectively discarding tuples or computing approximate summaries. This is the case, for example, of Aurora [5] and TelegraphCQ [6].

Existing proposals in both environments incur a key limitation: they reduce the flexibility of the system by either restricting the queries to a pre-defined set of well-known metrics or limiting them to a small set of operators with known and constant cost. This reduces significantly the possible applications and scenarios where these systems can be used. Our approach differs from those solutions in that it does not require any explicit knowledge of the traffic queries, allowing them to perform any arbitrary computation on the incoming traffic stream. Instead, queries are treated as black boxes with variable input traffic, output results, and arbitrary resource consumption.

III. System Overview

The CoMo platform (Continuous Monitoring) [7] is an open source passive network monitoring system designed for capturing network traffic at high speeds and computing user-defined queries in real time. CoMo follows a modular approach where users can easily define traffic queries as plug-in modules written in the C programming language. In order to provide developers with maximum flexibility, CoMo does not impose any restriction on the data structures a query can use, allowing them to perform any arbitrary computation on the incoming traffic stream. As a consequence, the platform does not have any explicit knowledge of the cost of each query. This flexibility makes the problem of managing the system resources particularly challenging. More details about the CoMo platform can be found in [7].

Given the bursty and unpredictable nature of network traffic, in order to prevent uncontrolled packet losses (which would have an unpredictable impact on the accuracy of the results), the system must anticipate overload situations and take load shedding decisions in advance. Since the platform considers all queries as black boxes, the only solution consists of measuring their resource requirements and learning their resource usage patterns from these external observations.

Without any knowledge about the queries, we infer their cost from the correlation between a large set of predefined traffic features and the actual CPU usage of each query. As we describe in Section IV.A, a traffic feature is simply a counter that describes a specific property of a sequence of packets. For example, potential features could be the number of packets in the sequence, the amount of bytes, the number of unique IP destination addresses, etc. The main advantage of the features we compute is that they are lightweight with a deterministic worst case computational cost.

The basic intuition behind our method comes from the empirical observation that the cost of a query is dominated by the overhead of maintaining the data structures needed to keep its state, which directly depends on different properties of the input traffic (i.e. the traffic features). For example, the cost of a query performing some sort of flow classification using a hash table will mainly depend on the number of new flows in the traffic, assuming that the cost of creating a new entry is much higher than the cost of updating an existing one. The main novelty of our approach is that it does not require any explicit knowledge of the queries. This way we preserve a high degree of flexibility in the monitoring system, increasing the potential applications and network scenarios where it can be used.

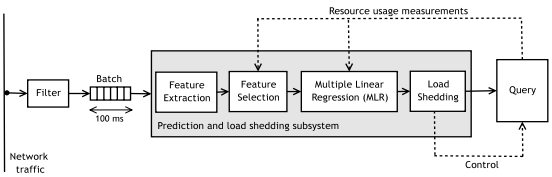


Fig. 2 Prediction and load shedding system

Fig. 2 presents the four components of our prediction and load shedding system, which are described in detail in the following sections. The CoMo monitoring system collects the traffic from the wire, filters the packets of interest and groups them in time-fixed batches (e.g. 100*ms*). Each batch is then analyzed by our system in order to predict the CPU cycles required by each query to process the entire batch, based on recent observations of the previous resource usage. If the prediction exceeds the system capacity, the load shedding subsystem discards a portion of the packets via packet or flow sampling to reduce the load of the system. Finally, the resulting batch is processed by each query and the actual CPU usage is measured to update the history maintained by the prediction subsystem. In order to measure the CPU usage, we use the *time-stamp counter* (TSC) as described in [1].

IV. Prediction Method

In this section we describe the three components of our online prediction method (i.e. *feature extraction*, *feature selection* and *multiple linear regression*) that constitute the core of our resource management system.

A. Feature Extraction

Traffic features are the input of our prediction method. Our goal is to extract a set of features that, with low overhead, provides enough information on the traffic characteristics that dominate the processing cost of a wide range of queries.

A feature that is too specific would add overhead without contributing to build a better resource usage model. Alternatively, a feature could provide valuable information but have a cost, in terms of CPU, comparable to the cost of a query, which is not desirable.

In our system, we use two kinds of counters. First, two simple counters: the number of packets and the number of captured bytes. These are of very low cost, but can explain a great portion of the cost of most queries. Second, we calculate a set of traffic aggregates that hold the count of unique occurrences of several combinations of the 5-tuple header fields in the traffic (e.g. unique destination IP addresses and ports). A detailed list of the features we compute on the traffic stream is available in [1].

A naive algorithm for calculating these traffic aggregates

would be very expensive in terms of both CPU time and memory space. Fortunately, the recent literature provides us with efficient approaches that can obtain approximate counts in linear time and bounded memory space [8]-[9]. In this work, we use multi-resolution bitmaps [8] to calculate traffic aggregates.

B. Feature Selection

The set of features presented in the previous section helps to explain the cost of many queries. However, not all the extracted features are relevant to each query. The feature selection stage selects the relevant features that exhibit correlation with the cost of the traffic queries. This has two beneficial outcomes: it reduces the noise in the prediction and it lightens the cost of building the prediction model.

Our feature selection algorithm is based on the fast correlation-based filter (FCBF) from [10]. Based on past observations of each query, this algorithm can discard both features that do not exhibit any correlation with the CPU usage (*irrelevant features*) and features that become linear combinations of other relevant features under a particular input traffic (*redundant features*), and therefore do not provide any new information to build the prediction model.

C. Multiple Linear Regression

The main thesis behind our resource management approach is that our simple traffic features provide a characterization of the incoming traffic that can be used to build a prediction model of the CPU usage of each query.

The system maintains a table with the previous values of each traffic feature together with the actual CPU usage of each query. The prediction subsystem uses this information to generate a prediction model of each query. It considers the selected features as predictors, whereas the CPU usage is the response variable (i.e. the variable we want to predict).

We experimentally observed that no single predictor provides enough information to explain in detail the cost of most queries. Instead, it is the combination of many features that can help to accurately predict the CPU usage of a query. Our system uses a multiple linear regression (MLR) to predict the response variable from several predictors.

The MLR studies the relationship between a response variable Y and p predictor variables X_1, X_2, …, X_p, and assumes that Y is a linear function of the predictor variables. The general form of a linear regression model with a history of n observations can be expressed as follows [11]:

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_p X_{pi} + \varepsilon_i, \quad i = 1,2,\ldots,n \tag{1}$$

In the previous expression, all Y_i variables correspond to previous observations of the response variable. The β_1 … β_p variables are known as the regression coefficients or the weight that each predictor variable has on the response variable. Using the ordinary least squares (OLS) procedure, the estimators b of the β coefficients are calculated so that the sum of squares of the residuals ε_i is minimized.

Note that the MLR assumes that the relationship between the cycles consumed by a query is in fact a linear combination of the traffic features. While in practice in our system the MLR provides accurate results, as we show in Section VI, other methods that can account for non-linear relationships between predictors and response variables constitute an important piece of future work.

V. Load Shedding

In this section, we present our approach for dynamically reducing the system load. In particular, our system controls the CPU cycles allocated to each query by dynamically modifying the sampling rate applied to each of them, based on the output of the prediction subsystem.

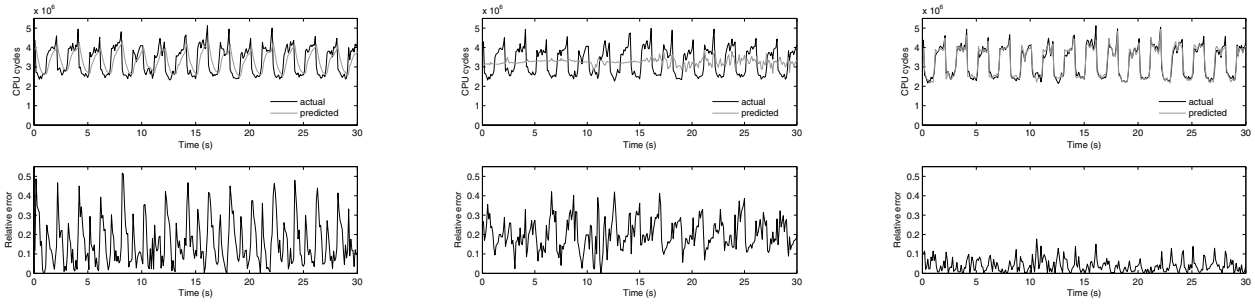
In our current implementation, we only support packet and flow sampling, and assume that each query selects at configuration time the option that would yield the best results. We implement packet sampling by randomly selecting packets in a batch with probability p (i.e. the *sampling rate*), while flow sampling randomly selects entire flows with probability p. To efficiently implement flow sampling, we compute a randomly chosen hash function [12] on the 5-tuple flow ID, which distributes the flows randomly. Then, if the hash value is smaller than $p \times (2^n\text{-}1)$, where n is the number of bits of the hash value, the packet is selected. Finally, the actual number of packets or flows can be simply estimated by multiplying the number of sampled packets or flows by the inverse of the sampling rate.

It is clear that there is a large set of imaginable queries that are not able to correctly estimate their unsampled output from sampled streams, even when flow sampling is used instead of packet sampling. For those queries, we plan to implement other load shedding mechanisms, such as computing different lightweight summaries of the input data streams [6].

A. Load Shedding Algorithm

Packet or flow sampling is applied only when the sum of the predictions for all queries (p*redicted_cycles*) exceeds an overall threshold (*available_cycles*), which is dynamically set according to the CPU cycles available in the system to process a batch. Since batches corresponds to fixed time bins (e.g. 100*ms* in our experiments), this threshold can be easily computed as *available_cycles = (0.1 × CPU_frequency) − overhead*, where *overhead* is the cost of our prediction method plus the cycles spent by other tasks carried out by CoMo, but not directly related to query processing (e.g. packet collection and memory management). This overhead is measured using the TSC. When the prediction exceeds the *available_cycles* threshold, the sampling rate to be applied is set in all queries to the ratio *available_cycles / predicted_cycles*. However, this ratio is only an estimate of the most appropriate sampling rate, because the CPU usage of a query can depend on several traffic features or on a different feature than the number of packets (or 5-tuple flows when flow sampling is used).

For this reason, before processing the sampled batch, we recompute the prediction in order to assure that the applied sampling rate is enough to shed excess of load. We repeat this process until the prediction is below the threshold or a fixed



(a) Exponential Weighted Moving Average (b) Simple Linear Regression (c) Multiple Linear Regression

Fig. 3 Prediction accuracy in the presence of Distributed Denial of Service Attacks

number of retries is exceeded. In [1] we also show how the sampling rate can be corrected to compensate for the prediction error observed in previous batches and to account for the space available in the system buffers.

VI. Robustness against Traffic Anomalies

Network monitoring systems are mostly needed in presence of unfriendly traffic mixes. The system may observe extreme traffic conditions when it is monitoring an ongoing denial of service attack, worm infection, or even an attack targeted to the measurement system itself. During these events, the query results, even if approximate, are extremely valuable to network operators. In this section, we evaluate the robustness of an actual implementation of our prediction and load shedding methods in the presence of anomalous traffic. In [1] we evaluate our load shedding scheme under normal traffic conditions.

A. Alternative Approaches

In this experiment, we compare the robustness of our prediction model against two well-known prediction techniques, namely the Exponentially Weighted Moving Average (EWMA) and the Simple Linear Regression (SLR), in the presence of traffic anomalies.

Exponentially Weighted Moving Average. EWMA is one of the most frequently applied time-series prediction techniques. It uses an exponentially decreasing weighted average of the past observations of a variable to predict its future values. EWMA can be written as:

$$\hat{Y}_{t+1} = \alpha Y_t + (1-\alpha)\hat{Y}_t \qquad (2)$$

where $\hat{Y}_{t+1}$ is the prediction for the instant $t+1$, which is computed as the weighted average between the real value of Y and its estimated value at the instant t, and α is the *weight*, also known as the *smoothing constant*. The results shown in this section consider the best prediction accuracy we obtained in most of our experiments, which corresponds to $\alpha = 0.3$.

Simple Linear Regression. SLR is a particular case of the multiple linear regression model where one single prediction variable is used. The SLR model can be written as:

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i \qquad i = 1, 2, \ldots, n \qquad (3)$$

where X is the prediction variable, β_0 is the intercept, β_1 is the unknown coefficient and ε_i are the residuals. As in the case of MLR, the estimation b for the unknown β is obtained by minimizing the sum of the squared errors. In the experiments presented in this section, we use the number of packets as predictor to compute the linear regression.

In order to test our prediction system (that drives the load shedding scheme), we injected synthetic anomalies in a 30-minute trace collected in November 2005 at a Gigabit Ethernet link that connects the Catalan and the Spanish National Research and Education Networks. The trace accounts for 103.67M packets, with an average rate of 360.46 Mbps and a peak rate of 483.28 Mbps.

We have generated many different types of attacks, such as simple volume-based denial of service attacks (i.e. an overwhelming number of packets destined to a single target), worm outbreaks (i.e. a large number of packets from many different sources and destinations while keeping fixed the destination port number) or attacks against our monitoring system (i.e. attacks that result in a highly variable and unpredictable workload for the system).

Fig. 3 shows the performance of each method in the presence of attacks targeted to the monitoring system. We injected in the trace a distributed denial of service attack with spoofed source IP addresses and ports, which goes idle every other second to generate a higher variable workload.

The figure shows the performance for a query (called *flows*) that aggregates the incoming packets into flows and reports the number of packets and bytes per flow. We chose this query, from those in the standard distribution of CoMo, because it is the most sensitive to this type of attacks.

In Fig. 3(c), we can see that MLR predictions track the actual CPU usage very closely, with errors around the 10% mark (with an average error of 4.77%). MLR can anticipate the increase in CPU cycles while EWMA (Fig. 3(a)) is always a little behind resulting in large oscillations in the prediction error. In the case of SLR (Fig. 3(b)), since the number of packets does not vary as much as the number of 5-tuple flows, the errors are more stable but persistently around 30% (it converges to the average cost between the anomalous and normal traffic).

We also generated other types of attacks that targeted other queries with similar results. For example, we generated an attack consisting of sending bursts of 1500 byte long packets for those queries that depend on the number of bytes (e.g. collecting a packet trace and pattern matching).

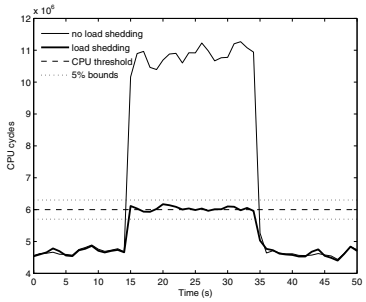


Fig. 4 CPU usage with and without load shedding

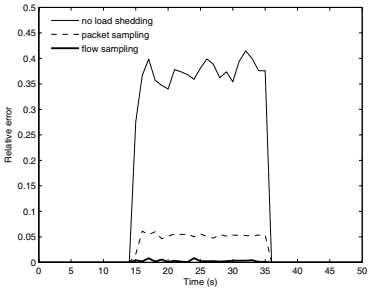


Fig. 5 Error in the query results with and without load shedding

B. *Impact of Overload Situations*

In the following experiment, we evaluate the impact of this kind of attacks on an actual implementation based on our prediction method. As an example, we consider a query that tracks the number of active (i.e. for which at least one packet was observed) 5-tuple flows in the packet stream and reports the count every measurement interval. We have implemented this query with a few simple modifications to the *flows* query used in the previous experiment.

In order to ease the exposition of the results, we manually set a limit (*available_cycles*) on the amount of CPU cycles available to the query for processing a batch (set to 6M cycles per batch). The sampling rate to be applied, when the CPU usage is above this threshold, is set to the ratio *available_cycles / predicted_ cycles*.

During 20 seconds we inject a burst of traffic corresponding to a SYN-flood attack with spoofed IP source addresses to force higher CPU usage.

Fig. 4 shows the evolution of the CPU usage during the anomaly with and without load shedding (with flow sampling) for the trace in our dataset. Without load shedding, the CPU usage increases from 4.5M to 11M cycles during the anomaly (assuming an infinite buffer that causes no packet drops). Instead, when load shedding is enabled, the CPU usage is well under control within a 5% margin of the target usage.

Fig. 5 shows the query accuracy during the anomaly. To estimate the error in the absence of load shedding, we emulate a system with a buffer of 200*ms* of traffic and 6M cycles available to process incoming traffic. If the CPU usage exceeds 6M cycles per batch, we assume that a queue of packets starts building up until the buffer is full and incoming packets are dropped without control. When load shedding is enabled, the error in the estimation of the number of flows using flow sampling is less than 1%, while when using packet sampling it is slightly larger than 5%. Without load shedding, the measurement error is in the 35-40% range.

VII. CONCLUSIONS AND FUTURE WORK

Robustness against overload situations is crucial for network monitoring systems, which must inevitably deal with arbitrary data and computations in a resource constrained environment. In this paper, we discussed the challenges involved in managing the resources of network monitoring systems and reviewed the design of an online predictive approach to cope with overload situations.

Our method is based on modeling the resource usage of arbitrary traffic queries in real-time by observing the characteristics of the input traffic. We use a dynamic set of traffic features to build a prediction model of the CPU usage. This model is used to accurately predict the CPU requirements of the system and dynamically select the highest sampling rate that prevents packet losses.

We implemented our method in the CoMo platform and evaluated its robustness in front of overload situations injecting synthetic traffic anomalies. Our results indicate that the system efficiently handles extreme overload situations and minimizes their impact on the accuracy of the results.

The technique presented in this work is intentionally simple to operate in real-time. We are currently working on allowing users to define utility functions for their queries to maximize the utility of the monitoring system. Another important piece of future work consists of extending our techniques to other system resources and using similar approaches to address the resource management problem in a distributed environment.

REFERENCES

[1] P. Barlet-Ros, G. Iannaccone, J. Sanjuàs-Cuxart, D. Amores-López, and J. Solé-Pareta, "Load shedding in network monitoring applications," in Proc. of USENIX Annual Technical Conference, 2007.

[2] K. Suh, Y. Guo, J. Kurose, and D. Towsley, "Locating network monitors: complexity, heuristics, and coverage," in Proc. of IEEE Infocom, 2005.

[3] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better NetFlow," in Proc. of ACM Sigcomm, 2004.

[4] K. Keys, D. Moore, and C. Estan, "A robust system for accurate real-time summaries of internet traffic," in Proc. of ACM Sigmetrics, 2005.

[5] N. Tatbul, U. Çetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker, "Load shedding in a data stream manager," in Proc. of Intl. Conf. on Very Large Data Bases, 2003.

[6] F. Reiss and J. M. Hellerstein, "Declarative network monitoring with an underprovisioned query processor," in Proc. of Intl. Conf. on Data Engineering, 2006.

[7] G. Iannaccone, "Fast prototyping of network data mining applications," in Proc. of Passive and Active Measurement Conference, 2006.

[8] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high-speed links," in Proc. of ACM Sigcomm Conf. on Internet Measurement, 2003.

[9] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in Proc. of Intl. Workshop on Randomization and Approximation Techniques, 2002.

[10] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in Proc. of Intl. Conf. on Machine Learning, 2003.

[11] W. R. Dillon and M. Goldstein, *Multivariate Analysis: Methods and Applications*, John Wiley and Sons, 1984.

[12] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," Jornal of Computer and System Sciences, vol. 18, num. 2, 1979.