Resource Usage Modeling for Network Monitoring Applications

Josep Sanjuàs-Cuxart and Pere Barlet-Ros
Universitat Politècnica de Catalunya
Barcelona, Spain
{jsanjuas,pbarlet}@ac.upc.edu

Abstract

Building robust network monitoring applications is hard given the unpredictable nature of network traffic and high, ever-increasing data rates. Traffic analysis systems must be designed with load shedding techniques in mind that can reduce the workload of a network monitoring system whilst gracefully degrading the accuracy of the results. We present a novel load shedding approach based on building a model of the resource consumption of monitoring applications and using it to prevent overload. The system extracts a set of features from the traffic in the form of counters, and measures the resource usage of the monitoring tasks. This information is used to build a multiple linear regression based model of the monitoring task. This model can be used to predict the resource usage of the monitoring tasks, and therefore to select the appropriate level of load shedding with great accuracy and in a fine-grained basis. We implement and deploy or system on a high-speed link of a large academic ISP. Our results show that our system predicts resource usage with errors below 5%, and that the predictions can be used to fully prevent uncontrolled packet loss.

1 Introduction

Passive network monitoring systems extract in real time a set of metrics from the traffic that traverses network links. The insight they provide on the network traffic is crucial for the operation of networks, and aids perform, among other tasks, troubleshooting, anomaly detection, capacity planning and traffic engineering.

The most important challenge that monitoring systems face is keeping up with the incoming traffic data rates, that is, to be able to process all the packets without loss of accuracy. This is hard to achieve, given the trend of continuous growth of network bandwidth. It is not sufficient for the system to perform above the average input traffic data rate, because it is prohibitively expensive to dimension system buffers to absorb sustained peaks. It is also important to note that preserving the accuracy of the monitoring tasks is especially important in the case of anomalies or extreme traffic mixes, as this is when the network operators value the most the reports from monitoring applications. It is therefore crucial to prevent uncontrolled packet loss under heavy load.

Several research proposals have addressed this challenge. The proposals fall in two broad categories. Firstly, some proposals consider a fixed set of well-known traffic metrics that the monitoring systems calculate and degrade the accuracy of the results in the presence of overload. Secondly, other proposals consider a system that can compute arbitrary metrics, which are called *queries* and which are defined using a declarative language, usually inspired on SQL, with a limited set of operators whose cost and selectivity is assumed to be well known. This makes the system aware of the cost of the computations it performs so that it can determine the right amount of load to shed. The proposals in both these sets incur a key limitation: they limit the utility of the monitoring systems by restricting either the metrics the system calculates or the computations it can perform to calculate them.

Our proposal differs from the solutions proposed in the literature in that it does not require any explicit knowledge about the queries and therefore does not restrict the kind of computations that the monitoring tasks are allowed to do. Instead, queries are treated as black boxes with an input traffic, output results, and a measurable resource consumption.

Each query maintains a set of data structures in order to calculate the output result. The resource consumption of a query is then devoted to maintaining this set of data structures. For example, to calculate the number of flows in a traffic stream, a query can use a hash table. Resources will be spent by the query to maintain the hash table, by either creating or updating entries. Our thesis is that the cost of the queries, which is dominated by the maintenance costs of its data struc-

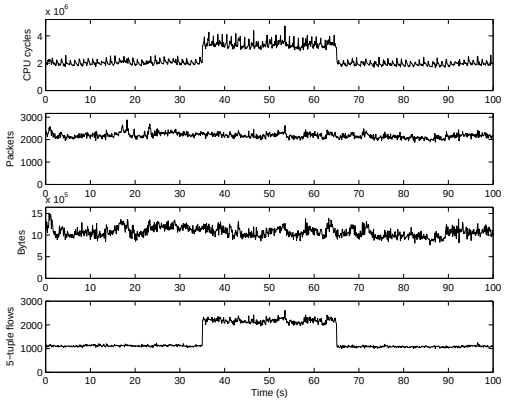


Figure 1. CPU usage of a query compared to the number of packets, bytes and flows in the traffic

tures, can be modeled and predicted by examining the characteristics of the input traffic.

In figure 1 we illustrate this point by comparing the cost in CPU cycles of a query to three characteristics of the input traffic. We have artificially introduced an anomaly in the input traffic that increases the number of flows in the traffic stream from seconds 35 to 65. It is clear that the cost of the query highly depends on the number of flows in the traffic, whereas it does not exhibit such a high degree of dependence with the other two features.

In this paper, we present a system that builds a model of the resource usage of each query. It extracts a set of lightweight metrics from the input traffic, and measures the CPU cycles that the queries utilize to process the traffic. Our system then uses the models to *anticipate* overload situations and immediately shed the necessary amount of load.

2 Related Work [1]

The design of mechanisms to handle overload situations is a classical problem in any real-time system design and several previous works have proposed solutions to the problem.

In the network monitoring space, NetFlow [4] is considered the state-of-the-art. In order to handle the large volumes of data exported and to reduce the load on the router it resorts to packet sampling. The sampling rate must be defined at configuration time, and to handle unexpected traffic scenarios network operators tend to set it to a low "safe" value (e.g., 1/100 or 1/1000 packets). Adaptive NetFlow [7] allows routers to dynamically tune the sampling rate to the memory consumption in order to maximize the accuracy given a specific incoming traffic mix. Keys et al. [11] extend the approach used in NetFlow by extracting and exporting a set of 12 traffic summaries that allow to answer a fixed number of common questions asked by network operators. They deal with extreme traffic conditions using adaptive sampling and memory-efficient counting algorithms. Our work differs from this approach in that we are not limited to a small set of known traffic summaries but instead we can handle arbitrary network data mining applications.

Several research proposals in the stream database literature are also very relevant to our work. The Aurora system [2] can process a large number of concurrent queries that are built out of a small set of operators. In Aurora, load shedding is achieved by inserting additional drop operators in the data flow of each query [14]. In order to find the proper location to insert the drop operators, [14] assumes explicit knowledge of the cost and selectivity of each operator in the data flow.

In [3, 13], the authors propose a system that applies approximate query processing techniques, instead of dropping records, to provide approximate and delay-bounded answers in presence of overload. Our work differs from these approaches in that we have no explicit knowledge on the query and therefore we cannot make any assumption on its cost or selectivity to know when it is the right time to drop records. Regarding the records to be dropped, we apply packet or flow sampling to reduce the load on the system, but other summarization techniques constitute an important piece of future work.

Our system is based on extracting features from the traffic streams with deterministic worst case time bounds. Several solutions have been proposed in the literature to this end. For example, counting the number of distinct items in a stream has been addressed in the past in [8, 1]. In this work we implemented the multi-resolution bitmap algorithms for counting flows proposed in [8].

3 System Overview

In this section we present a description of the system, with emphasis on the architecture of the monitoring platform, and the load shedding methodology. We also discuss the main challenges behind the CPU cy-

[1] Due to time constraints before the submission deadline, this section has been taken from "Load Shedding in Network Monitoring Applications", by Pere Barlet-Ros, Gianluca Iannaccone, Josep Sanjuàs-Cuxart, Diego Amores-López and Josep Solé-Pareta. To appear in USENIX'07.

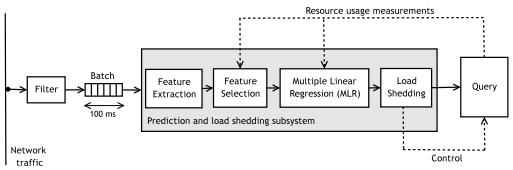


Figure 2. System overview

cles measurement, the traffic feature extraction mechanisms and the prediction subsystem.

As previously discussed, our approach to load shedding has a fundamental benefit over the previous proposals: it treats queries as black boxes and does not require knowledge about the computations that queries perform or their cost. To show the advantages of our technique, we have chosen the CoMo network monitoring system to implement our load shedding scheme. In the interest of space, we omit the details about the architecture and design of the CoMo network monitoring system which are not required to present our load shedding techniques. The interested reader can find details on the architecture of CoMo in [9].

In CoMo, queries are implemented in the form of plug-in modules written in the C programming language, while the core system performs all the tasks common to any monitoring application. Previous declarative language language-based approaches to load shedding were not applicable to CoMo, so this system serves well the purpose of illustrating our scheme.

Figure 2 shows the components and the data flow of our system. First, packets are read from the wire and the packets of interest are filtered. The packets are then grouped in batches of packets of $100ms$. Batches enter the prediction and load shedding subsystem, which consists of four phases:

- Feature Extraction. In this phase the traffic contained in the batch is characterized by extracting a set of features. Each feature is a counter that maps a characteristic of the traffic. The detailed list of traffic features can be found in section 3.3.

- Feature Selection. The system maintains a history of the traffic features and actual cost of each query. In this phase, the system uses this history to determine what features help explain the cost of each query. Only each query's relevant features are taken into account in the next phase.

- Multiple Linear Regression. In this step, the system builds a prediction model for each query using the history and taking into account the selected features, based on the Multiple Linear Regression. The system uses the models and traffic features to emit a prediction of the CPU usage of each query.

- Load Shedding. By comparing the predicted CPU load with the available cycles, in this stage the system can precisely determine whether to shed load, and how much load to shed.

Finally, batches are passed to all the queries, which calculate the metrics the users are interested in. The CPU usage of each query is monitored, and the measurements are used to update the recorded history of each query.

Table 1. Queries used in the experimental evaluation

Name	Description
application	Port-based application classification
flows	Per-flow counters
high-watermark	High watermark of link utilization
link-count	Traffic load
popular destinations	Per-flow counters for top destination IPs
string search	Finds a sequence of bytes in the payloads
trace	Full-payload trace collection

3.1 Queries

For validation purposes, in this work we have considered the set of queries that are available in the standard distribution of CoMo. In table 1 we present the queries with a brief description. We believe this set of queries is representative enough of the typical workloads and applications of network monitoring systems. Moreover, they use different data structures that lead to diverse CPU utilization patterns: aggregated counters, hash tables, linked lists.

3.2 Measurement of CPU cycles

Our scheme requires accurate measurement of the CPU usage of the queries to generate accurate models of the resource usage of the queries. Fine-grained measurement of CPU usage is not a trivial task for two main reasons: the lack of support from operating systems and the amount of noise in the measurements.

Lack of operating system support. System calls such as `gettimeofday()` do not provide enough resolution for the accurate measurements required in this work. For this reason, in this work we use the timestamp counter (TSC) register present in the x86 family of processors. The TSC is incremented once per each

CPU cycle, and therefore provides extremely high resolution. The TSC can be read from user-space with the `rdtsc` assembly instruction.

Measurement noise. The noise comes from three main factors. First, instruction reordering. In order to maximize its performance, modern CPUs aggressively reorders instructions according to their dependencies. Since the `rdtsc` instruction has no data dependencies on other instructions, it is a good candidate for reordering. This is an issue we tackle with the use of serializing instructions [10] before and after each call to the `rdtsc` instruction.

Second, context switches. Since our monitoring system has no control on the operating system's scheduler, the monitoring tasks may be scheduled out in favor of other tasks. This heavily affects the CPU measurements. In order to minimize the impact of this source of noise, we run the monitoring system with maximum priority and we monitor context switches with the `getrusage()` system call. Whenever a context switch is reported, we discard the current measurements.

Third, competition for the bus. System activity such as DMA disk accesses or memory accesses from processes running in other CPUs in SMP systems are a source of noise. However, in practice this source of noise is negligible and, as seen in the evaluation section, our system performs well under such circumstances.

3.3 Traffic feature extraction

Traffic features are extracted from the traffic in order to help in the prediction stage. Our goal is to calculate a set of features that, with low overhead, provides information on the characteristics of the traffic that helps explain the cost of a wide range of queries. A feature too specific would add overhead without contributing to build better resource usage models. 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 kind of counters. First, two simple counters: the number of packets and the number of captured bytes. These are of very low cost and, especially for the former, explain a great portion of the cost of most queries. Secondly, we calculate a set of traffic aggregates. These counters hold the count of unique occurrences of header fields or combinations of fields in the traffic. The actual traffic aggregates we compute are described in table 2.

A naive algorithm for calculating traffic aggregates would be very expensive in terms of both CPU time and memory space. The recent literature provides efficient approaches that provide approximate counts of these aggregates in linear time and bounded memory space [8, 1]. In this work, we use multi-resolution bitmaps [8] to calculate traffic aggregates. We dimension the bitmaps to obtain an error below 5%.

Table 2. Traffic aggregates

1	src-ip
2	dst-ip
3	protocol
4	<src-ip, dst-ip>
5	<src-port, proto>
6	<dst-port, proto>
7	<src-ip, src-port, proto>
8	<dst-ip, dst-port, proto>
9	<src-port, dst-port, proto>
10	<src-ip, dst-ip, src-port, dst-port, proto>

3.4 Feature Selection

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

Our feature selection algorithm is based on the fast correlation-based filter (FCBF) from [16]. This algorithm can discard both the features that do not exhibit any correlation with the CPU usage of queries, and the features which are correlated with other relevant features and therefore do not contribute useful information in order to build the prediction models.

3.5 Prediction methodology

The core of our approach to load shedding is that the traffic features provide a characterization of the traffic which can be used to build a model of the CPU usage of the queries.

The system maintains a table with the history of traffic features and the actual cycle 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 measured CPU cycles are referred to as the *response variable*.

No single variable can be considered to explain in detail the cost of a query. It is the combination of many features that provides enough information to predict the CPU cycles of queries. Our system uses the

well-known multiple linear regression (MLR) to build a model that can 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, $\ldots$, 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 of the variables can be expressed as follows [5]:

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

In the previous expression, all Y_i variables correspond to observations of the response variable. The $\beta_1 \ldots \beta_p$ 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) method, the estimators b_0 $\ldots$ b_p for $\beta_0 \ldots \beta_p$ are calculated so that the residuals ε_i are minimized.

In our system we use the single value decomposition method [12] to compute the OLS, which, in spite of being more computationally expensive than others, yields the best results approximation of the regression coefficients for over- or under-determined systems. The OLS method relies on several statistical properties which must be fulfilled. In the interest of space we do not discuss these issues in this paper.

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 good results, as shown in the system evaluation in section 4, investigating on algorithms and methods that can account for non-linear relationships between predictors and response variables constitutes an important piece of future work.

3.6 The load shedding subsystem

The load shedding subsystem is in charge of shedding the excess load to accommodate the demands of the queries with the available CPU cycles. The fundamental questions a load shedder must answer are (*i*) when to shed load, (*ii*) where to shed the load and (*iii*) how much load to shed. In our system, this translates to deciding whether a batch requires load shedding, which queries to shed the load from, and how many cycles must be saved by shedding load.

The first and second questions are answered using the modeling predictions for each query and comparing the sum of the predictions to the available processing cycles per batch. In order for the system to know how much available cycles it has, we have instrumented CoMo with tools to measure the CPU usage of the core system, besides measuring the usage of queries. Intuitively, a global input traffic sampling rate that sheds the required load can be calculated as follows:

$$sampling\_rate = \min\left(1, \frac{available\_cycles}{predicted\_cycles}\right)$$

In order to shed the required load, we apply this global sampling rate to each query. We acknowledge that this approach is rather unsophisticated. However, in this paper we focus in resource usage modeling as a tool to predict the resource usage of each query. Therefore, we aim for a simplified approach to load shedding that serves the purpose of validating the usefulness of resource usage modeling for load shedding. We disregard more complex load shedding techniques which can be devised to account for priorities of queries, or apply more shedding to the most cycle-demanding queries.

4 Evaluation

In this section we present an evaluation of our system. We focus on two major points. First, that the prediction models are accurate. Second, that the system is robust enough to perform load shedding on real-time, by analyzing live traffic on a high bandwidth link.

4.1 Environment

Our testbed scenario consists of two commodity PCs, with Intel Pentium IV processors that run at 3GHz. Each computer is equipped with an Endace DAG 4.3GE card [6] network card for high-speed collection of packets. Both computers receive an exact copy of the traffic of a link of the Catalan Research and Education Network, an academic ISP that connects the Catalan universities to the Internet. In order to ensure that both computers receive exact copies of the traffic, an optical splitter has been used to replicate the traffic on the link.

4.2 Prediction Accuracy

To evaluate the accuracy of the models, we collect two traces from our testbed. We run a modified version of our system that does not perform load shedding, but just performs the predictions, and runs the queries. This allows for comparing the predictions with the actual CPU usage of queries. Instead of running from live traffic, and for repeatability purposes, we collect two traces from the link. In the first trace we store

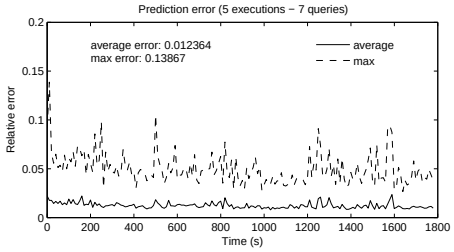


Figure 3. MLR error (trace with payloads)

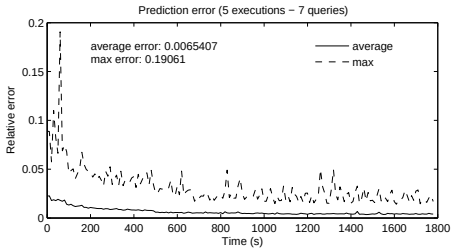


Figure 4. MLR error (trace without payloads)

an exact copy of the traffic as seen on the wire. In the second trace we do not store the payloads, but only the packet headers.

In figure 3 we present a time-series of the prediction error for the trace with packet payloads. It can be seen that the average error across all queries is around 1%, while the error peaks around 13% but normally is around 5%.

On the other hand, as seen in figure 4 the system performs slightly better on the trace without payloads. The average error is well below 1%, and while the error peaks at 19%, the maximum error is during a big portion of the time series below 5%.

We speculate that the system performs better on the trace without payloads due to the decreased interference of disk activity with the CPU measurements.

4.3 Evaluation of the full system

In this section, we evaluate the system running on live traffic. We implement and compare two additional variants of our load shedding strategy. We name the approach we present in this paper *predictive*. The *original* variant of the system performs no load shedding, i.e. is subject to packet loss on overloads. The *reactive* variant is not predictive. Instead, it selects the sampling rate based on the measurements of the previous batch, which is an approach similar to that of SEDA [15].

In order to be able to evaluate the error of queries, we implement a mechanism to notify queries of the

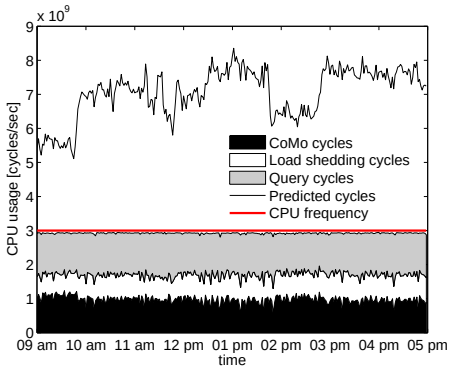


Figure 6. CPU usage after load shedding (stacked) and estimated CPU usage (predictive approach)

sampling rate applied to the input traffic, so that queries can estimate their unsampled output. Also, queries are allowed to choose between packet sampling and flow-wise sampling at configuration time.

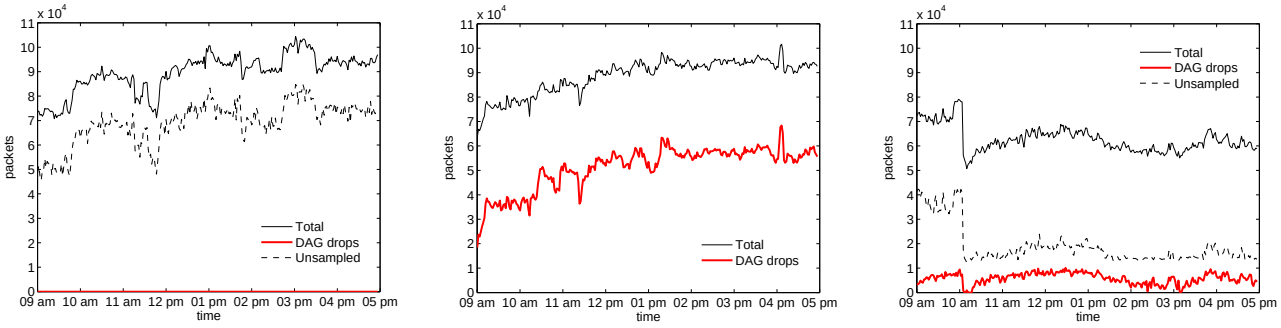
Figure 5 compares the amount of packet drops in each approach and shows the superiority of the approach presented in this paper. The load imposed by the input traffic is roughly equivalent to twice as much as the system can handle. Unsurprisingly, the *original* approach drops an amount of packets in the order of 50%. The *reactive* variant of the system performs much better than the *original*, as it is able to reduce the amount of packet loss by selecting low sampling rates.

In contrast, our technique scores *zero* packet loss during an 8-hour long experiment under equally adverse traffic load. It is also important to note that not only it prevents packet loss, which is easily achievable by imposing conservatively low sampling rates, but it also does so while applying high sampling rates. This is highly desirable as higher sampling rates lead to reduced errors in the results of queries.

4.4 Overhead

Figure 6 shows a breakdown of the CPU cycles of the system depending on the task they are used in. The CoMo cycles represent the cycles that CoMo requires to do the tasks common to all queries and which can not be attributed to the technique we present in this paper, which correspond to the Load shedding techniques in the figure.

The overhead our technique introduces is not negligible, but it is reasonably low given the advantages it provides to the operation of the system. The bulk



(a) *Predictive* load shedding (b) *Original* CoMo (c) *Reactive* load shedding

Figure 5. Link load and packet drops during the evaluation of each load shedding method

of the overhead corresponds to the feature extraction phase. Therefore, accelerating this phase would greatly reduce this overhead. The algorithms used in this step are suitable for hardware implementation. Another viable alternative to reduce their cost would be to decrease the accuracy of their approximate results. We have experienced great variations in the overhead by altering the size of the bitmaps. An interesting piece of work would be to analyze the associated trade-offs, as reducing the accuracy of the feature extraction algorithms would save cycles but would in turn degrade the accuracy of the prediction subsystem.

5 Conclusions and future work

Load shedding is crucial for network monitoring systems, which must inevitably be able to cope with overload during their operation. In this work, we have investigated a predictive approach to load shedding that contrasts with the reactive approaches that can be found in the literature.

Our method is based on real-time modeling of network monitoring applications by observing the characteristics of the input traffic. We extract a set of features from the traffic and, for each query, select the relevant ones to build a prediction model of the CPU usage. This models can be used to accurately predict the CPU requirements of queries. This accurate, short-term prediction permits selecting the highest sampling rate that prevents packet losses.

We implemented our load shedding system on the CoMo network monitoring system. We show that our predictive scheme is able to prevent packet loss on a 8 hour long execution under high traffic data rates. On the contrary, alternative, reactive approaches fail to contain CPU usage and suffer from packet loss.

During the paper we have identified several ideas for future work. Reducing the overhead of our technique would make a nice improvement to the system. Also, it would be interesting to investigate on prediction models besides the MLR, which can account for non-linear relationships between traffic features and the cost of the queries.

The load shedding subsystem presented in this work is intentionally simple. We are working on providing users with the possibility of defining utility functions for their queries, so that the system can maximize its own aggregate utility.

Another important piece of future work is to extend the techniques to other system resources besides the CPU power such as the system memory.

Acknowledgements

The work described in this paper has been done by Pere Barlet-Ros, Gianluca Iannaccone, Josep Sanjuàs-Cuxart, Diego Amores-López and Josep Solé-Pareta.

This work was funded by a University Research Grant awarded by the Intel Research Council, and by the Spanish Ministry of Education (MEC) under contract TEC2005-08051-C03-01 (CATARO project). Authors would also like to thank the Supercomputing Center of Catalonia (CESCA) for allowing them to collect the packet traces used in this work.

References

[1] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proceedings of RANDOM*, 2002.

[2] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Monitoring streams - a new class of data management applications. In *VLDB*, 2002.

[3] S. Chandrasekaran et al. TelegraphCQ: Continuous dataflow processing of an uncertain world. In *Proceedings of CIDR*, 2003.
[4] Cisco Systems. NetFlow services and applications. White Paper, 2000.
[5] W. R. Dillon and M. Goldstein. *Multivariate Analysis: Methods and Applications*. John Wiley and Sons, 1984.
[6] Endace. http://www.endace.com.
[7] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better NetFlow. In *Proceedings of ACM Sigcomm*, Aug. 2004.
[8] C. Estan, G. Varghese, and M. Fisk. Bitmap algorithms for counting active flows on high speed links. In *Proceedings of ACM IMC*, 2003.
[9] G. Iannaccone, C. Diot, D. McAuley, A. Moore, I. Pratt, and L. Rizzo. The CoMo white paper. Technical report, Intel Research, Sept. 2004.
[10] Intel. *The IA-32 Intel Architecture Software Developer's Manual*. 2006.
[11] K. Keys, D. Moore, and C. Estan. A robust system for accurate real-time summaries of internet traffic. In *Proceedings of SIGMETRICS*, 2005.
[12] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1992.
[13] F. R. Reiss and J. M. Hellerstein. Declarative network monitoring with an underprovisioned query processor. In *ICDE*, 2006.
[14] N. Tatbul, U. Çetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *Proceedings of VLDB*, 2003.
[15] M. Welsh, D. E. Culler, and E. A. Brewer. SEDA: An architecture for well-conditioned, scalable internet services. In *Proc. of ACM Symposium on Operating System Principles*, pages 230–243, 2001.
[16] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of ICML*, 2003.