Efficient Storage and Processing of High-Volume Network Monitoring Data

Giuseppe Aceto, Alessio Botta, *Member, IEEE*, Antonio Pescapé, *Senior Member, IEEE*, and Cedric Westphal, *Senior Member, IEEE*

***Abstract*—Monitoring modern networks involves storing and transferring huge amounts of data. To cope with this problem, in this paper we propose a technique that allows to transform the measurement data in a representation format meeting two main objectives at the same time. Firstly, it allows to perform a number of operations directly on the transformed data with a controlled loss of accuracy, thanks to the mathematical framework it is based on. Secondly, the new representation has a small memory footprint, allowing to reduce the space needed for data storage and the time needed for data transfer. To validate our technique, we perform an analysis of its performance in terms of accuracy and memory footprint. The results show that the transformed data closely approximates the original data (within 5% relative error) while achieving a compression ratio of 20%; storage footprint can also be gradually reduced towards the one of the state-of-the-art compression tools, such as bzip2, if higher approximation is allowed. Finally, a sensibility analysis show that technique allows to trade-off the accuracy on different input fields so to accommodate for specific application needs, while a scalability analysis indicates that the technique scales with input size spanning up to three orders of magnitude.**

***Index Terms*—Network monitoring, network measurements, traffic analysis, monitoring data compression.**

I. Introduction

TELECOM operators and their partners have to constantly monitor the network for a number of tasks like billing, management, provisioning, dimensioning, service offerings, etc.. Tasks such as billing require the analysis of the data in near real-time, while others are performed a posteriori on historical data sets. For instance, network provisioning is performed on a set of statistical indicators calculated over the last months or years. For these tasks monitoring infrastructures have been presented such as [2], relying on data reduction techniques to keep the amount of data manageable. Similar issues have been early addressed by the networking research community, e.g. [3] but technology evolution has worsened them, requiring more "aggressive" lossy approaches such as adaptive shedding of input data [4]. On the other hand, the availability of network monitoring data has allowed for more complex analyses such as behavioral pattern mining [5], highly valuable for both content providers and communication infrastructure managers.

These tasks often require to save the monitoring data; this implies storing for a long time a huge amount of information. For instance, a telecom operator willing to analyze service access patterns could easily be confronted with large data sets to be transmitted from monitoring points to processing sites, and accumulated for several months. Another example is provided by companies working in the field of Internet advertisement (ads). They have to make decisions based upon a high volume of data really fast to target ads and compute a bid price for them. The placement decision relies on patterns identified in the data logs (such as users with similar behavior as the user to present with an ad embedded in the web page he is looking at). The placement engines looks for a match if the presented ads would be clicked on with some high-enough likelihood. One possible solution resorts to state-of-the-art compression algorithms, in order to keep the storage footprint - and the transmission burden - manageable; but this approach has the drawback of needing a decompression stage (and then space for the uncompressed data) before any further processing. Operating on a log of a large number of users prohibit using compressed solution in real time, as the speed to present the ads is the key performance factor.

In order to cope with this issue, we propose a technique that (i) is efficient in space utilization, and (ii) provides the possibility to perform a class of operations directly on the compressed data. The technique we present trades off accuracy for reduced memory footprint and computational complexity for postprocessing, allowing for different levels of approximation or compression, according to the needs of the intended application. In a previous paper [1], we introduced the basics of this technique and a preliminary evaluation of its performance. In this paper we present the technique in details, including also various improvements we made recently. Moreover, we perform an in-depth performance analysis in terms of accuracy and memory footprint using different traces from different network scenarios. The results show that the transformed data closely approximates the original data while the compression ratio is close to that of the state-of-the-art compression tools, such as *bzip2*. Thanks to the new control parameters introduced, we also perform an analysis of two important aspects: sensitivity and scalability. The analysis of the sensitivity to the input parameters shows that it is possible to trade-off the accuracy on different input fields, which allows to cope with the requirements of different kinds of applications. The analysis performed varying the size of the input data allows to assess the scalability of the technique to input sizes spanning three orders of magnitude.

Manuscript received December 13, 2011; revised June 29, 2012; accepted January 4, 2013. The associate editor coordinating the review of this paper and approving it for publication was J. Sventek.

Authors are listed in alphabetical order. G. Aceto, A. Botta, and A. Pescapé are with the University of Napoli Federico II, Italy (e-mail: {giuseppe.aceto, a.botta, pescape}@unina.it).

C. Westphal is with Huawei Innovation Center, USA (e-mail: cedric.westphal@huawei.com). This work was partially conducted while he was at Docomo Innovation in Palo Alto, CA. Preliminary results within this framework have been presented in [1].

Digital Object Identifier 10.1109/TNSM.2013.13.110215

This paper is organized as follows. In Sec. II, we provide detailed information about our technique.Its possible applications are reported in Sec. III, illustrating a range of computations allowed on the compressed format. Sec. IV contains a deep experimental evaluation on different real data traces, captured in different environments. An analysis of the scalability to the log size and the sensitivity to the input parameters is reported in Sec. V. Finally, we discuss the related work in Sec. VII and draw conclusions in Sec. VIII.

II. The Technique

The technique we propose produces a spatially efficient data representation that allows approximated computations in network monitoring logs, with no need for decompression stage. In order to ease the description of the technique, of its characteristics, and the challenges it must deal with, we consider a specific format of monitoring log, but the technique can be applied to more complex monitoring data. The log format considered, analogous to flow-level traffic traces such as NetFlow ones [6], is constituted of records, each comprising four fields: *timestamp*, *source IP*, *destination URL*, and *load*. Each record represents a single HTTP session originated by a source IP to retrieve the given URL[1], and the amount of data that has been exchanged. This format represents an example of data commonly logged by operators.

To be able to perform computations directly on the representation, we apply a technique which converts the log in numerical matrices, where each row/column is a vector of real numbers. As shown in Fig. 1, our technique comprises two main phases. The first one is called pre-processing phase (II.A in the figure), and provides as output a fully numeric matrix representation of the original data. The second phase is called factorization phase (II.B in the figure), and it transforms the numeric matrix into a couple of sparse matrices which approximate our original data, while having a smaller memory footprint. In the following sections, we detail the two phases. Fig. 1 shows a block diagram of all the phases and suphases, using the same names as the related sections.

A. Preprocessing

1) Sessions splitting: When dealing with a continuous stream of monitoring data, there is the necessity to split it in parts (*chunks*) with a finite number of sessions in order to apply the subsequent mapping and processing. This can be performed in several ways, either by considering the possibility to have overlapping between subsequent chunks or forcing all of them to be disjoint, and allowing for fixed- or varying-size chunks according to different stopping criteria. The chunk size and the splitting policy affect i) the size of each compact representation chunk; ii) the data visible to the approximation algorithm, and thus both the compression efficiency and the approximation error; iii) the scope of the operations that can be performed directly on the representation (see Sec. III), that can be applied intra-chunk and inter-chunks. Moreover, according to the intended application, the splitting can be performed before or after the *URL filtering* (described in the following), with different consequences. For the analysis conducted in this paper, a fixed *split size* parameter is used to control the number of sessions allowed for each run of the approximation algorithm; this processing is done before the *URL filtering*. The *split size* parameter has been used for the scalability analysis reported in Sec. V-B: as this parameter controls the size of the input data and is considered in the first stage of the preprocessing phase, it allows to assess the scalability of the technique to the log size.

2) URL filtering: As in some applications singular events may be of no importance, the proposed technique provides the possibility to control the presence of HTTP sessions with rare servers. This is done by means of a *URL filtering threshold* parameter, defined as the minimum number of occurrences a URL must exhibit in the log in order to be considered. Such kind of filtering reduces the number of uniques URLs, but also affects some sources (all sources communicating with an under-the-threshold URL are discarded). As explained in Sec. IV, we performed an analysis of the effects of the filtering threshold, and the experiments reported in Sec. IV and Sec. V consider the results of this analysis.

3) Label Coding: The values in the field *URL* (destination URLs) are stored as a list of unique elements; URLs are ranked by number of occurrences, and mapped to an integer equal to their rank order: the higher the frequency, the higher the numerical label assigned to them. In a typical encoding, a shorter code is assigned to more frequent items. However, in our case, small values might be changed to zero when attempting to find a sparse representation and we would like to preserve the values of the most frequently accessed items. Even if source IPs could be represented as their numerical value, this value conveys little meaning, and presents a strongly uneven distribution. For this reason, we treat IPs as non-numerical values to be processed similarly to URL field. Therefore the values in the field *source IP* are stored as a list with unique elements: the label corresponding to a source IP is given by its position in the list, and referred to as *srcID*. The *load* field is already numerical in nature: the value in bytes from the log file is referred to as *Bytes* in the following. It is worth noting that while the matrix factorization has to be performed for each chunk of input data, the mapping processing can be done once and updated incrementally, simply adding the new codes (e.g. by means of a hashing function). This way the size of mapping affects the total size of the representation within a percentage that decreases with the growth of the number of sessions, further improving the scalability of the technique.

4) Normalization: The fields *srcID*, *Bytes* and *URLcode* are scaled to $[0, 1]$. For *srcID* this is done by dividing the code (ranking) by the total number of *srcID*s. The same is done for *URLcode*. Due to the high variance of values in *Bytes* field, base-2 logarithm of the value is taken, and divided by its maximum over the trace.

5) Weighting: According to the specific application of the technique, different degrees of approximation can be tolerated on the different fields: at this stage, it is possible to tune the relative importance of the fields by multiplying each of them by a *weighting* factor. Given an order for the fields, the array of their weighting factors defines the *weighting vector*, which constitutes an input parameter for the technique. This parameter can be exploited for both the initial configuration, and a per-chunk optimization. In the preliminary configuration

[1]We consider a general domain as a URL, without including arguments or subpaths within a domain.

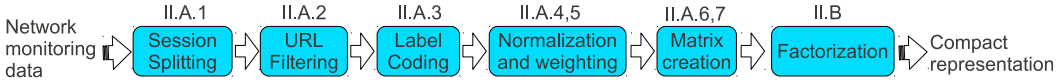


Fig. 1: Block diagram of the presented technique, with references to sections describing the steps.

phase, performed only once, the value of this parameter is to be defined in order to adapt the technique to the specific input data format (the nature of the different fields, their range, and the approximation error tolerated). Given a sample chunk of input data, an automated process to infer the convenient value for the *weighting vector* can be envisioned: e.g. we can use tools from the Operations Research field, using the requisites on accuracy and compression efficiency as constraints or as goal functions according to the intended application, or machine learning techniques. This is left as a future work. If the optimization is performed on a per-chunk basis, the local structure of data can be exploited for fine-tuning, and better overall performance is expected. A valuable property for this optimization is that, differently from the *session split* process, the *weighting* one does not affect the scope of computations even if applied on a per-chunk basis (and thus with a different value of the weight vector associated to different chunks). In fact, as it will be clarified in Sec. II-B, thanks to the mathematical properties of the factorization algorithm it is possible to apply linear transforms (and reverse them) to the compressed format as if they were applied to the input data (with an approximation error); therefore multiplying the original data by the weighting vector can be reversed in the computations performed directly on the representation (for a formal proof see Sec. III-A). An analysis of the effect of weighting is reported in Sec. V-A.

6) Timestamp: The timestamps are considered implicitly using the sequence number of the sessions, which allows to not include them in the matrix described in the following. We can easily calculate the maximum error we can have with this approximation. Let Δ be the average inter-arrival time of sessions within the observed time window τ. Thus, $\Delta = \tau/N$ where N is the number of sessions in the observed time window and τ is the width of the time window, that is affected by the *split size* parameter. The timestamp of entry k is replaced by:

$$\tilde{t}_k = t_1 + (k-1) * \Delta$$

where t_1 is the timestamp of the first session on the time window. Let a_i be the arrival time of session $i+1$ with respect to session i. If the number if session is high, a_i can be approximated with a Poisson distributed random variable with mean $E[a] = \Delta$ (and thus variance Δ as well). The actual arrival time for the k-th connection is thus $t_1 + \sum_{i=1}^{k-1} a_i$. The error in the timestamp is: $e_k = \sum_{i=1}^{k-1} a_i - (k-1)\Delta$. For the central limit theorem, e_k is distributed according to a normal distribution $\mathcal{N}(0, \sqrt{k}\Delta)$. Since $\Delta = \tau/N$ and $k \leq N$, the error is upper bounded by $\mathcal{N}(0, \tau/\sqrt{N})$, and it goes to zero as the number of connections grows within a time window τ. Considering as a typical scenario the monitoring on a gateway link, the system under consideration sees a number of sessions easily in order of millions per hour, thus the assumption that N is large is reasonable. In our ongoing work we are also considering other representations for the timestamps.

7) Matrix format: As the last step of the preprocessing phase we build P, which is an $M \times N$ matrix, representing N sessions (with N equal to the *split size* parameter), each of which with M dimensions, depending on the number of fields (we described four fields in the input data format above, but we could consider other flow parameters as well). In this representation, each column of the input matrix is a vector $(srcID; URLcode; Bytes)^T$ where the time is implicitly represented by column index, namely the i^{th} column holds values of the i^{th} entry of the input file.

B. Factorization

In order to store P efficiently, we want to use a matrix $\hat{P}$ such that: $\hat{P}$ approximates P, and $\hat{P} = TC$, where $T(M \times K)$ and $C(K \times N)$ are two matrices with high sparsity and thus requiring a small amount of memory for their storage. K is a parameter that is optimized when T and C are derived. K can be larger than M if this decreases the number of non-zero components in C. As our objective is to trade off computational complexity vs accuracy, we model our approximation as the minimization problem:

$$\min_{T,C} ||P - TC||_2^2 + \lambda(||T||_0 + ||C||_0) \tag{1}$$

where $||.||_0$ is the ℓ_0 norm that counts the number of non-zero elements of its argument, and λ is a Lagrangian multiplier which is specified by the user (we call λ granularity parameter because, as will be clear in the following, it controls the approximation error). T is the pattern basis and C a matrix of coefficients. Namely, a column vector $\hat{p}_k$ from $\hat{P}$ can be expressed as a decomposition along the basis vectors t_i in T:

$$\hat{p}_k = \sum_{i=1}^{K} c_k(i) t_i \tag{2}$$

T can thus be used to identify patterns in P and answer queries about patterns in the logged data. Note that since the optimization problem (1) takes into account the number of non-zero elements of T and C, it yields sparse results, thus reducing the storage requirement and also the computational complexity required by operations on the compressed representation. In more details, to answer a specific query that can be put in the form $Y = PX$, one can use $\hat{P}$ instead of P and solve $\hat{Y} = \hat{P}X = TCX$. Remember that $||T||_0$ and $||C||_0$ are minimized by construction. Therefore, computing the product CX requires at most $||C||_0$ multiplications of coefficients. Similarly, computing $Y = T(CX)$ requires no more than $||T||_0$ multiplications. Thus, the complexity of answering a query that can be put in the form $Y = PX$ is equal to $||T||_0 + ||C||_0$ operations.

In order to compute T and C, we use the technique proposed by Zujovic et al. [7] in the context of pattern matching algorithms (applied to query-by-example image re-

trieval)[2]. Assuming that T and all the elements of C are given, except $c_i(k)$. Following [7], the approximation modeled as (1) becomes the minimization:

$$||p_i - \sum_{l \neq k} c_i(l)t_l - c_i(k)t_k||_2^2 + \lambda||c_i(k)||_0 \quad (3)$$

Defining $e_{i,k}$ as the residual $p_i - \sum_{l \neq k} c_i(l)t_l$, the minimization becomes:

$$\min_{c_i(k)} ||e_{i,k} - c_i(k)t_k||_2^2 + \lambda||c_i(k)||_0 \quad (4)$$

The first term is minimized by finding $c_i(k)$ such that $c_i(k)t_k$ is the orthogonal projection of $e_{i,k}$ onto t_k. The second term takes value 0 or λ respectively if $c_i(k)$ is null or not. Solving jointly yields:

$$c_i(k) = \begin{cases} \frac{e_{i,k}t_k^T}{||t_k||_2^2} & \text{if } \frac{|e_{i,k}t_k^T|}{||t_k||_2^2} > \sqrt{\lambda} \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Since finding $c_i(k)$ in (5) only involves P, T and $c_i(l)$, then one can solve at the same time for the whole row $c_j(k)$, $j = 1, \ldots, M$. This defines the row optimization procedure $opt\_row(C|k, T, P, \lambda)$ [7]. Solving the initial problem (1) then becomes an iteration of this procedure, first applied on the rows of C ($opt\_row(C|k, T, P, \lambda)$) then on the columns of T (as (3) can be trivially mapped to a column optimization by taking the transpose, i.e. solving $opt\_row(T^T|k, C^T, P^T, \lambda)$). This two-step iteration is repeated until convergence of the matrices T and C, namely when the changes between two iterations are smaller than some ϵ. This constitutes the procedure $TC\_iterate(T, C|P, \lambda)$. The process is initiated with T and C equal to zero. The optimization also finds the proper value of K by looking at the largest eigenvalue of $E = P - TC$ and its associated eigenvector v. Define u to be a M-dimensional vector of 1s. If applying $TC\_iterate(v, u^T|E, \lambda)$ yields a non-zero value, then the output of the procedure can be added as a new column in T and a matching row in C, and K is thus incremented by one. This constitutes the $expand(T, C|P, \lambda)$ procedure. Both procedures $TC\_iterate(T, C|P, \lambda)$ and $expand(T, C|P, \lambda)$ are alternated until all K, T and C converge.

It is important to underline that the computations required for all the other phases are negligible with respect to the ones required for the factorization. Also, the other stages can be performed on the fly, while for the factorization it is necessary to have the complete log file before starting. The performance of the factorization stage are analyzed in detail in Sec. VI.

III. Applications

The technique proposed in this paper allows to answer a range of queries directly working on the new representation format. For instance, one can answer any max-k session query to find the k largest sessions in the log file. This can be solved by finding the k largest value of the $1 \times N$ row of $\hat{P}$ that corresponds to the load. One can similarly find the total usage of a specific *srcID*, by summing all values of *Bytes* $\hat{P}(3, i)$ for which $\hat{P}(1, i)$ = *srcID*. The matrix C points to which patterns in T the user calls upon. Thus similar users have similar coefficient in C, and can be identified by observing this sparse matrix. Conversely, the underlying matrix of patterns T embeds some overall behavior of the system and can be used to identify abnormal usage. In particular, if after computing T over some period of time Δ at regular intervals, one sees dramatic changes in the composition of T, say $\min_\pi ||T(t_2) - \pi T(t_1)||_2 > \gamma$ where π is a column permutation and γ a threshold, then it might point to some abnormal behavior in the system and call for some investigation. It is worth noting that the *split size* parameter (Sec. II-A1) affects both the set of users on which it is possible to perform direct comparison for similar behavior (the ones caught in the chunk), and the set of behaviors considered characteristic of the system (ending in building up the T matrix).

As a preliminary, we observe the following *convergence condition*: we note that (1) contains two terms, which are the ℓ_2 norm of the difference between $P - TC$, and the number of non-zero terms in T and C multiplied by the Lagrangian λ. From (5) we see that these two terms are of the same order of magnitude: a non-zero component is added by the algorithm only if the cost λ of adding this non-zero value to C or T is made up by the reduction in the difference $||P - TC||_2^2$.

If the data in P corresponds to observations drawn from an alphabet of K vectors $v_1, \ldots, v_K$, then T will have KM non-null components (namely the coordinates of the vectors $v_1, ..., v_K$) and C will have N non-null components (namely, N column vectors with a 1 in the position corresponding to which of the v_i is observed in the i-th row of P).

The second term of (1) can thus be approximated by $\lambda(MK + N)$, and the difference $P - TC$ satisfies:

$$||P - TC||_2^2 = \xi$$
$$||P - TC||_2^2 + \lambda(||T||_0 + ||C||_0) = \xi + \lambda(MK + N) \quad (6)$$

where, according to the convergence condition described above, $\xi \simeq \lambda(MK + N)$.

A. Linear queries

Since T and C are derived to minimize (1), we can write $P = TC + E$, where E is a matrix representing the distortion between P and its TC factorization. Any linear query on the data which corresponds to a linear equation $Y = PX$ thus can be answered by $\tilde{Y} = TCX$. The answer $\tilde{Y}$ will differ from Y by EX, and thus the error will be less than $||EX|| \leq ||E||||X||$. Since E is a function of λ, one can assess the accuracy of the answer based upon λ. In most practical cases, E will be Gaussian distributed noise and one can assess the probability that $\tilde{Y}$ provides a satisfactory answer using estimation theory and (6).

B. Detection of new patterns

Assume again that the observed values in P are drawn from an alphabet of K vectors $v_1, \ldots, v_K$. We are interested in detecting a change in the composition of the observed data, which means that a new vector v_{K+1} would be inserted in the mix. The vector v_{K+1} corresponds to a new or abnormal behavior in the data. Assume that n of the vectors in P are v_{K+1} and the remaining $N - n$ are drawn from the normal range of behavior $v_1, \ldots, v_K$. We have to consider two cases.

[2]The theoretical framework lies in the field of dictionary design for sparse representation, and is derived from [8].

In the first case, the algorithm accommodates the new pattern without changing T, and in the second, it does by adding a new pattern vector to T.

1) Constant-size T: In this scenario, T is still composed of K patterns. With no loss of generality, we assume that $v_1, \ldots, v_K$ remain the patterns in T and thus, v_{K+1} will be expressed by the algorithm as a component in the hyperplane defined by $v_1, \ldots, v_K$ and potentially an orthogonal component e_{K+1}. Assuming $v_{K+1} = e_{K+1} + \sum_{j=1}^{K} a_j v_j$, (1) yields two terms: the first term $||P - TC||_2^2$ is increased by $||e_{K+1}||_2^2$ by each one of the n new observations in P drawn from v_{K+1}. The second term $\lambda(||T||_0 + ||C||_0)$ is increased by $\lambda(K-1)n$ as each column in C now has to contain the $(a_1, \ldots, a_K)$ coordinates instead of the single non-zero value pointing to the observed vector v_i. There are n such columns in C.

In this case, the appearance of the n new observations drawn from v_{K+1} increases the minimization objective $||P - TC||_2^2 + \lambda(||T||_0 + ||C||_0)$ such that:

$$\begin{aligned} &||P - TC||_2^2 + \lambda(||T||_0 + ||C||_0) = \\ &= \xi + \lambda(MK + N) + n\left(||e_{K+1}||_2^2 + \lambda(K-1)\right) \end{aligned} \tag{7}$$

2) Increased-size T: The second possible case is that the appearance of the new observations triggers an increase in K, and a new pattern is added to the matrix T. In this case, the minimization objective $||P - TC||_2^2 + \lambda(||T||_0 + ||C||_0)$ is that of (6) with K replaced by $K + 1$.

$$\begin{aligned} ||P - TC||_2^2 + \lambda(||T||_0 + ||C||_0) &= \\ &= \xi + \lambda\left(M(K+1) + N\right) \\ &= \xi + \lambda(MK + N) + \lambda M \end{aligned} \tag{8}$$

The minimization objective will choose between (7) and (8), based on which objective is lower. Namely, it will choose the second scenario if

$$n\left(||e_{K+1}||_2^2 + \lambda(K-1)\right) \geq \lambda M \tag{9}$$

This is satisfied for $n \geq M/(K-1)$ and thus a new pattern will be easily detected by the increase in the dimension of the matrix T for $n \geq M/(K-1)$. For the values of M and K we consider, this means a new pattern would be detected almost immediately under the assumption that the observations are drawn from a finite alphabet.

IV. Experimental Evaluation

To evaluate the performance of our technique, we have considered its efficiency in terms of memory footprint reduction (*Compression Ratio*) and different consequences of the approximation (*Bytes error*, *ID error*, *URL error*). For the evaluation of the memory footprint, the compression ratio is compared with the output of the general purpose compression utility *bzip2*, employing the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. The reported results are obtained by varying the value of λ, that controls the amount of distortion allowed in the approximated factorization algorithm (the higher the value of λ, the higher the tolerated approximation, but also the more sparsity induced in the representation matrix): λ is varied in the set $\{0.001, 0.0025, 0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1\}$. The analysis are conducted with a fixed URL threshold. We performed an analysis of the effects of the filtering threshold, and found that 5 is the maximum value for which the accuracy is not significantly affected: all presented results refer to this value of URL threshold. This value is used for the analysis reported in the following sections.

TABLE I: Characteristics of traffic traces.

	LAB	MAWI
sessions	26965	105310
duration (s)	3599	899
URLs	1771	5926
IDs	110	12129

A. Data set

We used two real traffic traces, one captured on August 2010 at the gateway of a research facility (named LAB), and the other (named MAWI) captured on May 2010 on "Samplepoint F", a trans-oceanic link between Japan and USA, from the MAWI project [9]. Both traces include only packets with transport level port (either source or destination) equal to 80 and provide anonymized source and destination addresses. The characteristics of the two traces are reported in Tab. I. The numbers reported in this table are related to traces before filtering. We recall that the data set has the format of a log file, each record of which represents a single HTTP session, and constitutes of four fields: *timestamp* (in UNIX epoch time, μs precision), *source ID*, *destination URL*, *load* (in bytes).

B. Results

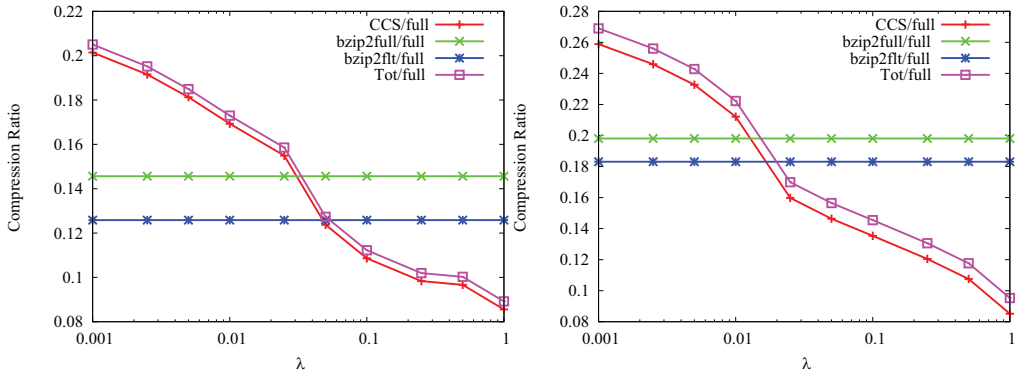
1) Compression Ratio: The total size of the data in the new representation format (we also call it compressed version), as well as the size of specific components, is compared against the size of the original data. The considered quantities are:

- CCS: size in bytes of the Compressed Column Sparse representation of the `C` matrix alone;
- Tot: sum of the size in bytes of: CCS, `T` matrix, bzip2-compressed ordered list of URLs, and bzip2-compressed ordered list of source IDs. These components, with a few metadata (namely the four integers representing the dimensions of the matrices and one float per field representing its scaling factor), are all we needed to rebuild the original data (URL-filtered and approximated);
- bzip2flt: size in bytes of the URL-filtered and bzip2-compressed version of the original data;
- bzip2full: size in bytes of the bzip2-compressed version of the original data with no filtering;
- full: size in bytes of the original data with no filtering.

The CCS component is calculated in bits as:

$$nnz \cdot (basesize + \lceil log_2(cols) \rceil) + \lceil log_2(nnz) \rceil \cdot (rows + 1)$$

where *nnz* is the number of non-zero elements of the matrix, *rows* and *cols* are the dimensions of the matrix, and $\lceil \cdot \rceil$ is the *ceiling* function. This value corresponds to the size occupancy of a sparse matrix, represented as Compressed Column Sparse (or *Compressed Sparse Column* [10]), where indexes are binary coded, and each element is represented with *basesize* bits. In the considered case, *basesize* is 32, *rows* and *cols* are the dimensions of matrix `C`, respectively K and



(a) LAB trace (b) MAWI trace

Fig. 2: Compression ratios.

N in the notation of the previous section. The matrix $\mathtt{T}$ is represented as $M \cdot K$ values of length *basesize* bits each.

Fig. 2a shows the compression ratio as a function of the granularity parameter λ for the LAB trace. As expected, increasing the approximation granularity, more sparsity is found in the factor matrix $\mathtt{C}$, and therefore the size occupancy for CCS representation decreases, causing the size of total representation to gracefully decrease.

Fig. 2b shows the compression ratio as a function of the granularity parameter λ for the MAWI trace. We can see a behavior analogous to the one shown for the LAB trace, with increasing sparsity found by higher values of λ. For $\lambda = 0.001$ the compression ratio Tot/full (i.e. the ratio between the total occupancy of the representation and the size of the original log file) is 0.21 for the LAB trace, and 0.27 for the MAWI trace, while for $\lambda = 1$ it reaches the value 0.08 for the first trace, and that of 0.1 for the second trace. The crossing point with the reference line (reporting the compression ratio of *bzip2*, namely bzip2full/full, i.e. the size of bzip2-compressed full log file divided by the size of the original file) is found for λ in-between 0.025 and 0.05 for the LAB trace, while for the MAWI trace it found for λ in between 0.01 and 0.025. This can be ascribed to the difference in the compression ratio achieved by *bzip2* in the two cases: for the LAB trace it amounts to about 0.15 while for the MAWI trace is about 0.20. A second reference line is also reported, namely bzip2flt/full, representing the ratio between the size of the URL-filtered, bzip2-compressed log and the size of the original log; for both the traces the gap between the two reference lines is of about 0.02, representing the relative gain in space for bzip2 compression due to the filtering step (described in Sec. II-A2).

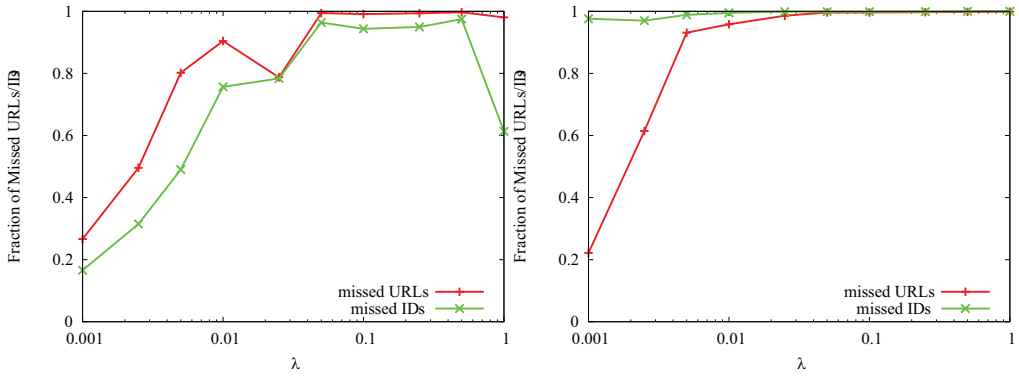
The ratios considered for the evaluation of compression efficiency, namely the size of CCS over the size of full log, and total size of representation over the size of full log, are separated by a gap that is constant with λ. The gap shows different values for the considered traces, being about 0.005 for the LAB trace, and more than 0.01 for the MAWI trace. This is consistent with the nature of such gap, deriving from the size of mapping of IDs and URLs onto the codes used for the representation (see Sec. II-A). The MAWI trace presents a higher number of unique IDs and URLs, and thus an increased gap between the size of the CCS representation and the total occupancy of the processed data. As noted in Sect. II-A3, the contribution of the mapping to the size of the representation can be set apart as it can be done once and just incrementally updated (with relatively negligible size increase), so for each chunk only the CCS representation contributes.

2) Error on ID and URLs decoding: Error on URL decoding is calculated as the ratio of entries with mistaken URLs versus the total number of entries. In order to calculate this value, a reconstruction of the original log is performed, using the approximated matrices and the index files. An URL is "mistaken" when the approximated index value, after the decoding, is closer to an index different from the original one. The same procedure is performed on source IDs.

The percentage of mistaken URLs and IDs for the LAB trace is shown in Fig. 3a. It can be seen that the errors increase until $\lambda = 0.01$, for $\lambda = 0.025$ there is a plateau[3], and then the error reaches almost 1. This is due to the rescaling phase that precedes the approximated matrix representation: in order to uniform the value span among data of different nature, the indexes are "compacted" so that for high values of approximation granularity, the distance between consecutive indexes becomes smaller than the allowed approximation error, eventually causing the decoding fault.

The percentage of mistaken URLs and IDs for the MAWI trace is shown in Fig. 3b. For the URLs we can observe a behavior similar to the one shown by IDs and URLs of the LAB trace: a quick raise from about 0.2 to almost 1, but reached for lower values of λ. The error ratio on IDs is significantly different, being close to 1 even for the smallest values of λ. The reason for this undesired performance is to be traced back to the number of unique IDs present in the MAWI trace, summing up to 1766:it is expected that when the representing codes are scaled between 0 and 1 the difference between adjacent codes becomes smaller than the allowed approximation error controlled by λ, leading to practical inability to decode. This problem can be overcome by exploiting the weighting phase (see Sec. II-A5), that counters the effect of scaling by multiplying a specific field by a weighting factor. An example of this is reported in Sec. V-A, where a weight is applied for the *IDs* field in order to improve the accuracy of its approximation.

[3]For $\lambda = 0.025$ the error slightly decreases. We argue that this happens because, for some approximation granularity values a slightly better sparse representation is achievable, especially for small traces, because λ is similar to a quantization parameter. Therefore, for small traces, the distribution of the samples may have an impact on the performance of the technique (i.e. changing the quantization intervals, samples may be closer to the borders and therefore experiment a smaller quantization error). In fact, in the larger trace (see Fig. 3b) this phenomenon does not happen.



(a) LAB trace (b) MAWI trace

Fig. 3: Fraction of Incorrect URLs and IDs.

3) Error on Bytes field: We consider the *Relative Error* on the *Bytes* field, defined as $\|\frac{x-\hat{x}}{x}\|$, where x is the true value, and $\hat{x}$ is the approximated value, affected by approximation error. The approximated value $\hat{x}$ is reconstructed, for each considered value of λ, by multiplying the matrices, rescaling by the inverse of the normalization factors and the weighting factors, and considering the *Byte* rows.

The extreme values, the quartiles and the mean of the *Relative Error* are shown in Fig. 4a for the LAB trace. Due to the wide variations of the Relative Error, the plot is in log-log scale. For y-axis a minimum of 10^{-3} has been set, as minimum values of relative error are always zero. We notice that the 75-percentile is always close to mean relative error (blue solid line), which increases with increasing λ, approximately varying as 10λ for values of λ in the decades $[0.001, 0.1]$ and lightly decreasing for λ in $[0.25, 1]$.

The trend shown by the *Relative Error* for the MAWI trace in Fig. 4b is analogous to the one exhibited by the LAB trace. When λ has small values (in between 0.001 and 0.25) the mean value is comparable to the values shown in the LAB trace, while for larger λ values the error reaches an almost constant value. The median in the MAWI trace performs comparatively better with regards to the other trace, starting at a much lower value (about 0.005 for $\lambda = 0.001$ instead of 0.01), and having smaller values for $\lambda < 0.1$. A reason for this can be drawn by considering the notable error on IDs for the same values of λ (see Fig. 3b): the new representation format trades-off accuracy between these two fields. Such trade-off is further analyzed in the sensitivity analysis performed in Sec. V-A, where it is shown that the improvement of error on *Bytes* field for small values of λ is reduced when accuracy on *IDs* field is enhanced by employing a weighting factor.

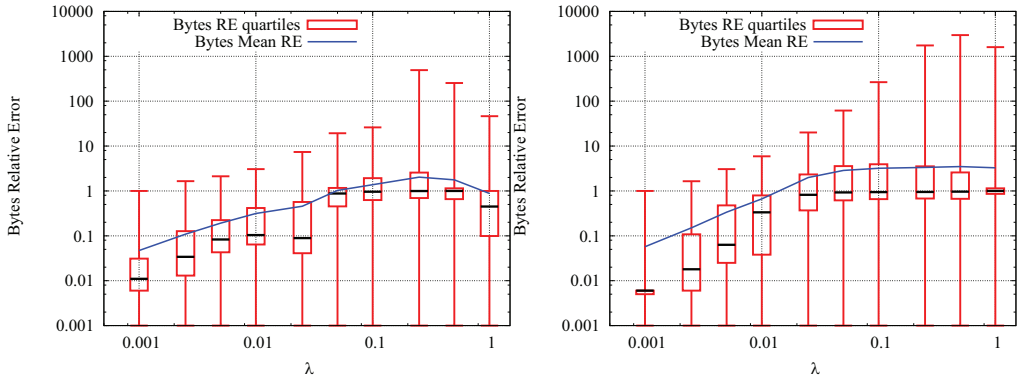
C. Discussion

From the experimental evaluation we verified that the technique is suitable for efficient space usage, as can be seen in Fig. 2a and 2b, where for fine-grained approximation ($\lambda = 10^{-3}$) the resulting space occupation becomes 21% of the space occupation of the original LAB trace and 27% of the space occupation of the original MAWI trace, and compression level equivalent to the one of *bzip2* is reached, for slightly coarser approximation (with λ equal to 0.06 and 0.02 for the LAB and MAWI traces respectively). This result, confirmed for two significantly different traces in terms of length and heterogeneity of IDs and URLs, is of notable value compared to *bzip2*: our format does not require decoding in order to perform a whole class of data mining and processing operations, as opposed to *bzip2* outcomes, that always need a decompression phase (and space for the decompressed data).

In Fig. 4a we can see that on the LAB trace the average relative error increases less than linearly for $\lambda \in \{10^{-3}, 25{\cdot}10^{-3}\}$, always close to the 75-percentile, while the median value is notably smaller, and less than or close to 10% of the true value even for coarse grained approximation. An analogous behavior can be detected for the MAWI trace: by observing Fig. 4b we find for the average a slow (in lin-log plot) increasing for $\lambda \in \{10^{-3}, 5 \cdot 10^{-2}\}$, closer to the 75-percentile for λ up to around 0.025; the median is always notably lower, and less than or close to the true value even for coarse grained approximation. This shows how the large part is barely affected, although some values can undergo a substantial and increasing approximation.

As shown in Fig. 3a and 3b, the fields that are mostly affected by approximation error are *IDs* and *URLs*, due to their nature of identifiers: in this case the decoding is either hit or miss, and "approximation" to the closer IDs is still a miss. A way to reduce this impact is provided by the tuning of the *weighting factor*, introduced in Sec. II-A5. In Sec. V-A we analyze how the *weighting factor* can help to mitigate the error on *IDs* and *URLs* fields and quantify the impact on the other fields. In general, the choice of allowed error, and thus, the setting of the control parameter λ, is dependent on the application. Values of λ exceeding 0.1, even if introducing significant approximation error, and higher variance for both traces (see Fig. 3a and 3b, 4a and 4b), achieve compression levels higher than the state-of-the-art reference (Fig. 2a and 2b), still retaining the possibility for direct processing without decompression. In applications where the exact storage of the value is not needed (e.g. in clustering), the graceful and controlled degrading of accuracy, selectively distributed among different monitoring fields, could be further exploited to gain even higher efficiency, as shown in the following.

Concluding, in contrast with lossless schemes such as *bzip2*, our technique allows to trade-off accuracy with storage space and processing time (as the amount of data stored increases). For this reason, we cannot say in advance what is the best



(a) LAB trace (b) MAWI trace

Fig. 4: Relative Error on *Bytes*: box plot shows minimum, 1^{st} and 3^{rd} quartile, and median; the line connects mean values.

operating condition (in terms of the different parameters that can be tuned), because it depends on the cost of an error, a false alarm, or failure in detecting an event. Some applications (e.g. operators identifying classes of behavior and thus the need for QoS guarantees for a number of users or services) can be more incline to tolerate losses in accuracy in order to have less data to store and analyze. Some others (e.g. billing) require data to be accurate, and will have to accommodate for more storage space and processing time for it. The control parameters of the technique have specifically been introduced to accommodate such different needs.

V. Sensitivity and Scalability Analysis

The technique we present can be tuned for specific applications or optimized for specific data characteristics. Besides the main parameter, λ, that deeply affects all the performance indexes of the technique and that has been analyzed and discussed in the previous sections, other secondary control parameters are of interest, namely the weighting vector (see Sec. II-A5) and the number of entries to be read from the input log file. This last parameter, being a control on the size of the input data, can also be used to perform a scalability analysis, as done in Sec. V-B. All the following analysis has been performed on the largest trace (MAWI).

A. Sensitivity to weighting factors

In this section we analyze the sensitivity of the technique to the weighting factors. More specifically, the purpose of introducing the weighting factors in the technique (i.e. allowing for a trade-off between accuracy on a specified field and compression efficiency or accuracy on other fields) is validated. We performed tests with several different weighting vectors. In the following, we report the most interesting results we obtained. In Fig. 5, the ratio of missed IDs and missed URLs is compared for two different weighting settings. In this figure, we indicate with a triple $w = (w_{ids}, w_{urls}, w_{bytes})$ the values of the weighting factors respectively of source ID codes, URL codes and Bytes value. We compare the results obtained with $w = (1, 1, 1)$, i.e. no selective weighting, and $w = (5, 1, 1)$, i.e. multiplying by a weight of 5 the *IDs* field. Note that the results obtained with $w = (1, 1, 1)$ are the same as those reported in Fig. 3. From Fig. 5 it can be seen that the decoding error on IDs is about 0.08 when the weighting

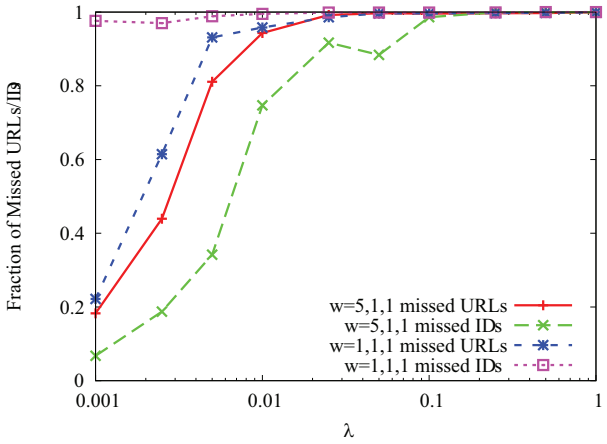


Fig. 5: Sensitivity to weighting factor: decoding error ratio for w=1,1,1 and w=5,1,1 with varying λ.

factor is equal to 5 and $\lambda = 0.001$, while in case of no weighting it was extremely high even for such fine-grained approximation. Moreover, it remains smaller than or close to 0.30 for $\lambda < 0.01$.

The trade-off for the accuracy improvement is evaluated in terms of loss of accuracy on the other fields (whose weighting factor is unchanged in the tests) and on the overall compression efficiency. Firstly we recall that for the *IDs* field, the improvement is verified for all the values of λ, with an enhancement that gradually becomes negligible for $\lambda > 0.1$. For the *URLs* field, we detect a small improvement for values of $\lambda \leq 0.01$, which becomes negligible for $\lambda > 0.025$. The reason for this behavior is discussed in the end of this section. A non monotonic behavior can be seen for the effect on the *Bytes* field, reported in Fig. 6. The mean value is only affected for $\lambda > 0.1$. For the median value, a small improvement (less than 0.01) for $\lambda \in [0.001, 0.005]$ can be observed.

In order to have an overall picture of the impact of weighting on all the fields, we report in Fig. 7 the differences between the errors on the fields for $w = (5, 1, 1)$ and the ones for $w = (1, 1, 1)$ (the more negative the difference, the better the effect of weighting). The relative error on *Bytes* (*BytesRE*), being a different kind of error (relative error instead of decoding error ratios), is reported on the secondary y axis, and due to high variability, a logarithmic scale is

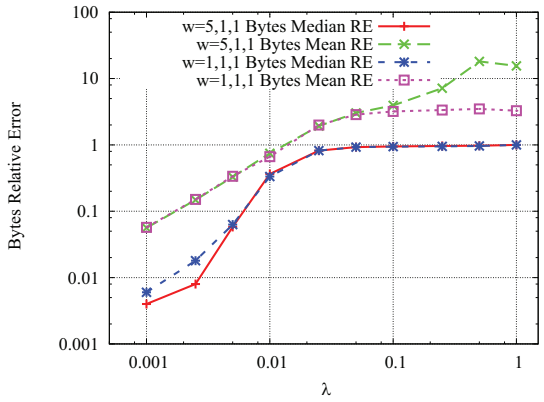


Fig. 6: Sensitivity to weighting factor: Relative Error on Bytes for w=1,1,1 and w=5,1,1 with varying λ.

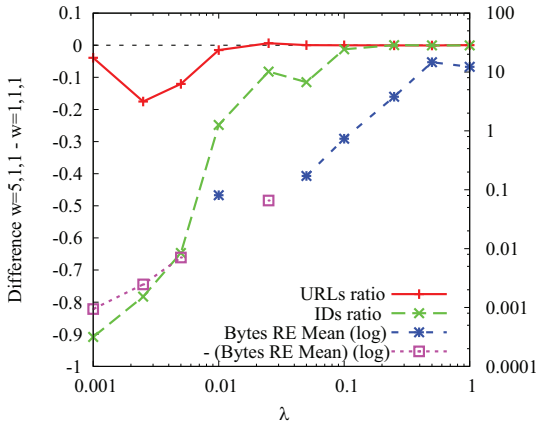


Fig. 7: Sensitivity to weighting factor: Differences between errors for w=5,1,1 and for w=1,1,1. The variations for bytes Relative Error are on logarithmic scale on the right y axis: negative values are separately reported with the sign changed.

used. Negative values are negated and drawn with a different symbol. The graph highlights that for $\lambda > 0.1$, the weighting has no more relevant effect on the accuracy of both *IDs* and *URLs* fields, while it adversely affects the accuracy for the *Bytes* field. Moreover, the mean of relative error on *Bytes* field for $\lambda \in [0.01, 0.025]$ shows a fluctuation of less than 0.08 between positive and negative variations. It is also noteworthy and unintuitive that for values of λ comprised in between 0.001 and 0.05 all the differences are negative, meaning that the errors decreased for all the fields: this is discussed in the end of the section.

The effect of the weighting factor on the overall compression efficiency is reported in Fig. 8, that is analogous to Fig. 2b to which we refer for the definitions of the index. As shown, there is actually a trade-off with accuracy, as the lines relative to $w = (5, 1, 1)$ are always above the ones relative to $w = (1, 1, 1)$, meaning that a loss of compression efficiency happens for all values of λ. More specifically, the most significant effect is found in the interval $\lambda \in [0.01, 0.1]$, where a decrease of compression of about 0.04 is detected. Outside this interval, the effects of weighting are almost constant with λ and small (less than 0.01). This shows also that, for λ warying in wide ranges, weighting does not heavily impact on the compression efficiency while still significantly improves the accuracy.

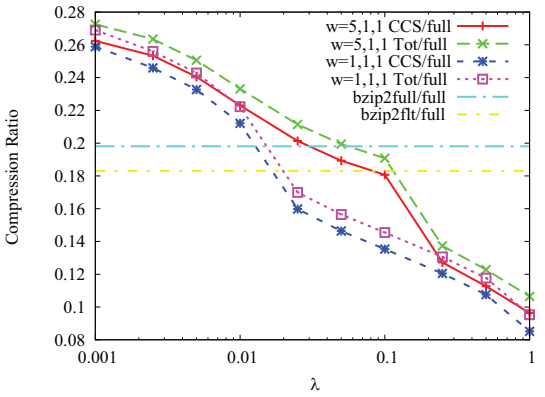


Fig. 8: Sensitivity to weighting factor: compression efficiency for w=1,1,1 and w=5,1,1 with varying λ.

The fact that for some values of λ the accuracy increases also for non-weighted fields (i.e. weight $= 1$) can be explained as follows. Increasing the weight (i.e. weight > 1) of a field has the (desired) effect of distributing the values of such field on a larger interval (i.e. spacing out the original values of this field). This implies that the approximation granularity (λ) is smaller with respect to the values of such field, and therefore, the approximation error can also be smaller. However, there is also another effect: the distribution of the values of one of the fields changes, leading the technique to find a different factorization for all the fields, which can also have different (better or worse) performance with respect to the original one. Clearly, this phenomenon becomes less evident on larger traces, as shown in Fig. 3b.

The whole sensitivity analysis shows that the weighting is largely beneficial for wide intervals of λ. The choice of the weighting vector is tightly related to the specific application scenario. In this paper we wanted to show the applicability of the technique to network monitoring data in a general case. Further analysis and optimization is left for specific case studies, that are out of the scope of this paper.

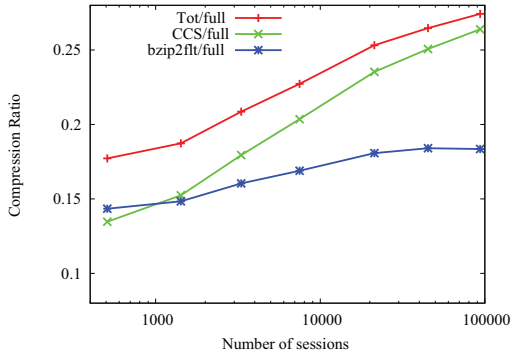
B. Scalability analysis

In order to evaluate the performance of the proposed technique in terms of compression and approximation error for a varying number of sessions, and to test the sensitivity of the performance on the *split size* parameter (see Sec. II-A1), we considered the largest trace (MAWI) and tested the technique on an increasing number of sessions. Throughout this evaluation we adopted a weighting vector $w = (5, 3, 1)$, chosen in order to have a better balance of errors between the *IDs* and *URLs* fields, without aiming at a specific application. Similar considerations can be done with the other values of the weighting vector. The results are reported in Fig. 9, 10 and 11 where the number of sessions remaining after filtering is reported on the x-axis. Tab. II reports the relation between the number of sessions allowed-in and the resulting number of sessions after filtering (see Sec. II-A2 for details). As regards the effect of filtering on considered traffic volume, we verified that the sessions selected from the full trace (accounting for $\sim 30\%$ of the total number of URLs) sum up to $\sim 84\%$ of the whole traffic volume.

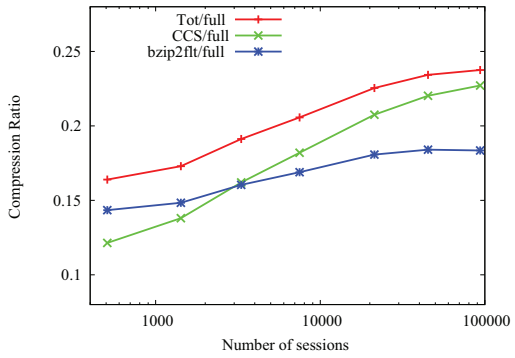
The variation of compression ratio is plotted against a varying number of sessions in Fig. 9a, 9b and 9c for λ equal

TABLE II: Number of sessions before and after filtering

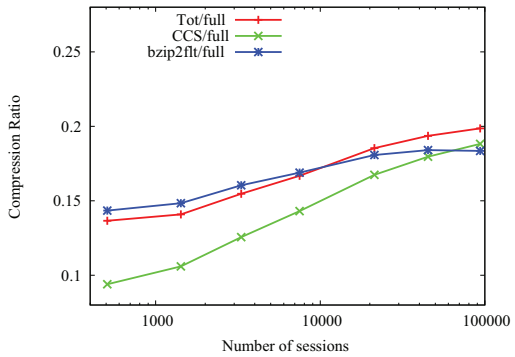
	After filtering		
Sessions	Sessions	URLs	IDs
1000	580	24	340
2500	1423	62	623
5000	3310	76	1133
10000	7486	167	1787
25000	21334	496	3228
50000	45136	1060	5398
100000	93649	1766	11245



(a) λ=0.001



(b) λ=0.01



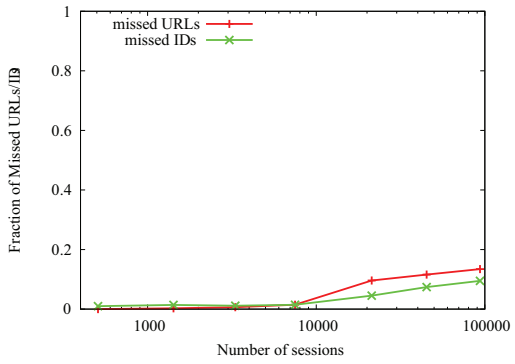
(c) λ=0.1

Fig. 9: Scalability analysis: compression ratios

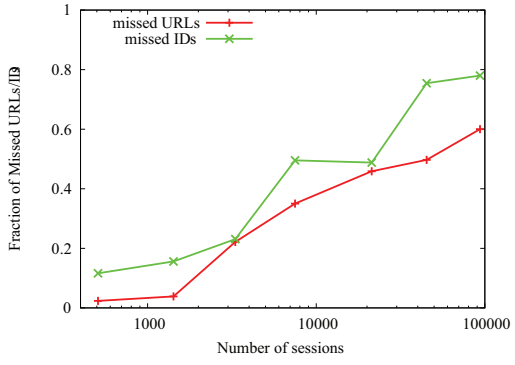
to 0.001, 0.01 and 0.1 respectively. The reported values are a suitable subset of the ones defined for Fig. 2b to which we refer for the description of the index. It can be noted that the ratio bzip2flt over the full size of original log is independent from λ, so its line (with the '*' symbol) is the same for the three plots, and is used as a reference. All the reported ratios (namely CCS representation, *bzip2* compression and whole representation) show an increasing trend for increasing number of sessions. Considering the plots in the sequence of increasing λ, the compression efficiency of our technique gets closer to the one of *bzip2*. For smaller number of sessions the CCS component is below *bzip2*, and the span of sessions for which this is true increases with increasing λ: the crossing point ranges from about 1500 sessions for $\lambda = 0.001$ up to 70000 sessions for $\lambda = 0.1$. For $\lambda = 0.1$ the compression efficiency of our technique becomes better than that of *bzip2*. This behavior is consistent on all the sessions and allows to generalize the results discussed in the previous section to a number of sessions spanning on three orders of magnitude. It also clearly indicates that the *split size* parameter (see Sec. II-A1) can be used to control the compression efficiency of the whole technique or, conversely, that it is possible to derive the optimal value of this parameter given a compression target. Fig. 9 shows also that for small λ the growth speed decreases with the number of sessions. For larger λ, the growth speed becomes smaller and smaller and stabilizes for large number of sessions. It is important to notice that the plot related to the CCS/full component (the green x in the figure), which is the fastest growing one, is limited by the one related to Tot/full, because Tot comprises CCS, as reported in the comment above. Therefore, Tot/full can be seen as upper bound to Tot/full.

The effect of the increasing number of sessions on the ratio of missed URLs and IDs is plotted in Fig. 10a, 10b, and 10c for λ equal to 0.001, 0.01 and 0.1 respectively. For all the values of λ, the increase of the numbers of sessions causes a growth of the missed ratio for both the *IDs* and *URLs* fields. More specifically for λ equal to 0.001 the effect is not relevant for numbers of sessions ranging from 500 up to 7486: at this level of approximation granularity, both IDs and $URLs$ are correctly decoded. Increasing the number of sessions, both ratios slightly increase but always remain under 0.1. The ratios gracefully raise (in a semilog plot) for *IDs* from about 0.1 when the number of sessions is equal to 580, up to 0.8 when the number of sessions is equal to 93649. A similar but smoother behavior is shown by the error on the *URLs* field, with values close to 0 and to 0.6 respectively. For λ equal to 0.01 the ratios gracefully raise for *IDs* from about 0.1, when the number of sessions is equal to 580, up to 0.8, when the number of sessions is equal to 93649. A similar but smoother behavior is shown by the error on the *URLs* field, with values close to 0 and to 0.6 respectively. For λ equal to 0.1 the missed ratio starts at about 0.6 for *IDs* and at about 0.25 for *URLs*, then they cross when the number of sessions is equal to 2500, and for 3000 or more sessions, the ratios show a very similar trend, smoothly getting close to 0.85.

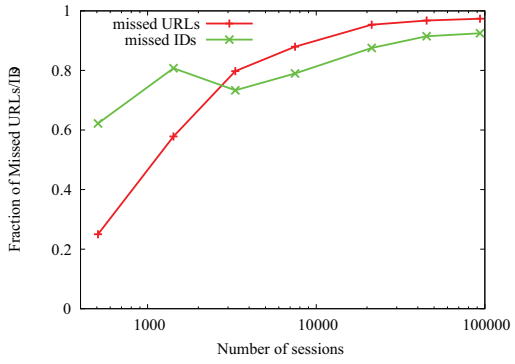
In general, the increase in the number of sessions leads to an increase of the number of unique IDs and unique URLs that have to be coded and "packed" in the $[0, 1]$ interval, decreasing the gap between two consecutive codes. When $\lambda = 0.001$ the accuracy is almost insensitive to the number of sessions. This allows to tune the *split size* parameter according to other targets (e.g. improving compression). In the case of $\lambda = 0.01$, the step-wise look of IDs ratio opposed to the smooth one of URLs can be explained by the fact that its coding gap is smaller than the one relative to URLs, and the error is kept smaller thanks to the higher weighting factor. In this case the use of weighting vector can be most effective, if the number of sessions is also taken into account. Finally it should be considered that such behavior is to be tracked back to the discreet nature of ID and URL codes. This constitutes



(a) λ=0.001

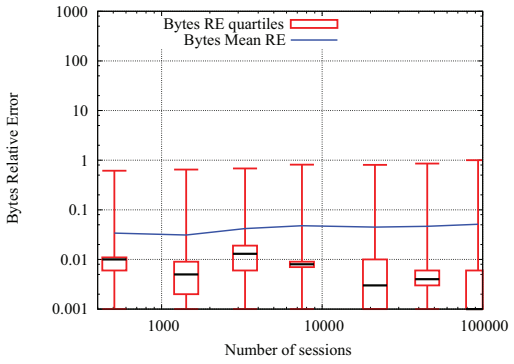


(b) λ=0.01

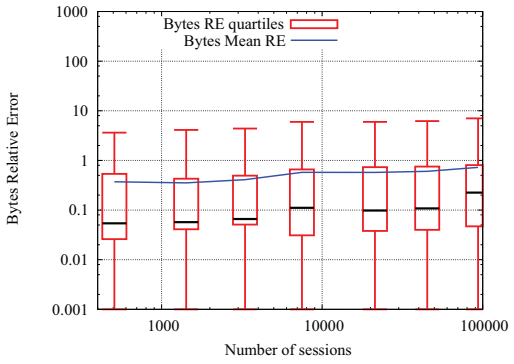


(c) λ=0.1

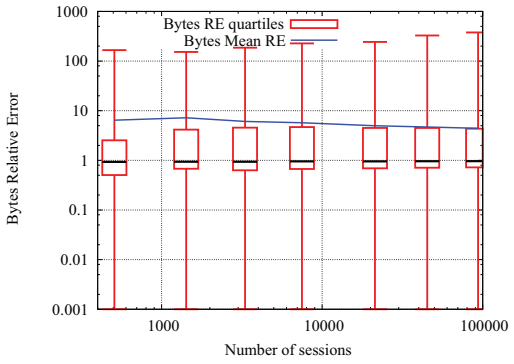
Fig. 10: Scalability analysis: missed IDs and URLs



(a) λ=0.001



(b) λ=0.01



(c) λ=0.1

Fig. 11: Scalability analysis: relative error on Bytes

a generic model for similar discrete fields, a representation choice for non-numerical fields in general.

The scalability results for *Bytes* field are reported in Fig. 11a, 11b, 11c for λ equal to 0.001, 0.01 and 0.1 respectively. For $\lambda = 0.001$, while the mean value is always smaller than 0.05, the quartiles oscillate around the value 0.01 for a number of sessions lower than 7486 and are smaller than 0.01 for increasing session number. As this corresponds to an increase in the missing ratio of IDs and URLs, we can explain this behavior as a trade-off in accuracy among these fields. Considering $\lambda = 0.01$, the quartiles show much more consistency across the sessions span, with the median slowly increasing from 0.05 up to 0.1 when the number of sessions is smaller than 10000, showing a plateau and then rising to 0.2 for the whole number of sessions available. The mean value is smaller than the 3^{rd} quartile. For $\lambda = 0.1$ a strong consistency is shown by all the quartiles, being almost constant for all the numbers of sessions; while the mean value shows a small decreasing trend from 0.9 to 0.5.

Notably, for each considered value of λ, the mean value of the relative error is almost constant on the whole span of session numbers, which is about three orders of magnitude. Considering a given number of sessions, relative errors on *bytes* increase with increasing values of λ, thus generalizing the analysis performed in former work [1]. The small sensitiveness of the mean of the relative error on *bytes* to the number of sessions allows the use of the *split size* parameter to freely control other parameters for the aim of optimizing other targets (i.e. the compression efficiency or the decoding efficiency on non-numerical fields). This analysis shows that the technique is scalable, especially from the point of view of the relative error on numerical fields.

VI. Performance Comparison

As explained in Sec. II, the factorization stage (see Fig. 1) is the one most demanding in terms of computational resources, and the time required by the other stages is negligible with respect to this one. In this section we compare the time required

TABLE III: Time required by our technique and *bzip2*, averaged on 10 runs.

Sessions	Our technique/*bzip2*	Our technique/*bunzip2*
2500	3.6	9.5
5000	3.0	10.1
10000	3.1	11.1
25000	2.6	11.3
50000	3.4	10.3
100000	3.7	7.8

for factorizing versus *bzip2* compression and decompression durations, using log files containing an increasing number of sessions. This is to show the benefits that our technique can provide in a real world scenario. The experiment has been performed on a workstation equipped with quad-core Intel® Xeon® E5540 2.53 GHz CPU, with 16 GB RAM. For this experiment, we used $\lambda = 0.025$ and $w = (5, 3, 1)$. As for the memory occupation, we verified that *bzip2* has an asymptotic behavior, limiting its maximum occupancy to about 30 MByte. The current implementation of our technique, instead, loads the entire matrix in memory. However, looking at the algorithm (Sec. II-B) we can see that it processes the matrix iteratively row by row. Therefore, we could severely limit the memory occupancy loading one row at a time. We left optimizations as future work.

Tab. III reports the ratio between the time required by our technique and that required by *bzip2* to compress the log file and *bunzip2* to uncompress it. As we can see, our technique takes $2.6 \rightarrow 3.7$ times more than *bzip2* and $7.8 \rightarrow 11.3$ times more than *bunzip2* for the same number of sessions. However, it is important to recall that *bzip2* requires decompressing the log file each and every time an analysis is required. For example, if we operate with *bzip2*/*bunzip2* on a log file with 100000 entries, we will need about $1/4$ of the time required by our technique to compress the file and about $1/8$ of such time for decompressing the file each time we have to perform an analysis. This means that our technique becomes convenient in terms of time if the logs have to be analyzed at least 6 times during their lifetime. Besides, it is important to note that using *bzip2*/*bunzip2* we also need a large storage space always free for decompressing the file before the analysis, and this can be prohibitive in case of very large log files. Finally, please note that the prototype code implementing our technique is a straightforward C realization of the algorithm, has no optimizations for memory and disk access, and is single threaded. Higher benefits are expected after the optimization of the code, which is out of the scope of this paper.

VII. Related Work

Several different approaches have been proposed to cope with the issues related to storing and processing huge amounts of data coming from network monitoring activities. An information theoretic framework is presented in [11], that within a network model, analyzes the information content of flow-level captures (using NetFlow-like format, which is analogous to the log file format we considered). By means of this modeling the authors derive the bounds for lossless compression of network traffic traces, resulting in about 20% theoretical compression ratio for the format they considered. [12] describes an off-line methodology complementing to sampling approaches for the reduction of data trace sizes while preserving mean, standard deviation and temporal structures (such as long range dependency and scaling behavior). It uses entropy to reduce the amount of data that has to be processed by traffic analysis tools. Entropy is used as input for an approach that ensures that sensible information needed to get an appropriate model of the network traffic is still present. Packets not needed for an appropriate model are dropped.

The problem of developing concise representations suitable for processing huge amount of data has been deeply analyzed by the Knowledge Discovery and Data Mining community. An example of efficient representation allowing for nearest-neighbour search is *iSAX*, presented in [13], where an indexing technique akin to extensible hashing is introduced for time series data. Though *iSAX* is not aimed at lossy compression and considers homogeneous data series, this technique and similar indexing ones could be adopted in the preprocessing stages to improve the overall performance of our technique.

Facing huge amounts of monitoring data, a possible strategy to enhance the performance of storing and processing is to apply lossless compression algorithms to a suitably preprocessed version of the data. An example of this approach can be found in [14], where a technique is presented that aims at compression improvement by reordering on high dimension data. By organizing data in *data matrices* in which each row is an occurrence of a multi-field values. The technique applies an optimization algorithm that changes row ordering to maximize the compression by *differential predictive coding*. The authors show that the optimal reordering leads to a Gaussian distribution for the prediction error in each column: in such a case the optimization algorithm is reduced to solving a Traveling Salesman Problem (TSP). The data matrix is allocated in a binary tree structure, and to allow application of the ordering algorithm on large data tables, a fast recursive partitioning TSP heuristics is used that has *O(NlogN)* time and *O(N)* space complexity. The *O(N)* space complexity does not allow to use this approach in data-stream scenarios, but could be considered for a multi-stage procedure, or applied to fixed-windowed time span, with expectable reduction in effectiveness. The authors report that the method is effective on moderately-large to large tables with intrinsic dimensionality below 20 and with attributes represented with low to moderate precision. Higher benefits may be obtained on data tables with correlated attributes and heterogeneous attribute types.

A similar approach, implying a preprocessing stage whose output is fed to a general-purpose compressor, is adopted in [15], where the input data shows a common basic format: ASCII text encoding, structured as a Character Separated Values database. The authors present a multi-layered approach, where the general-purpose compression algorithm (third stage) is prepended with a *string substitution stage* on subsequent couples of lines (resembling the backward reference coding of LZ77 [16]); they also exploit ASCII codes unused in the text (e.g. unprintable ones) as control characters for the decoding algorithm. Common approaches share a common scheme. The first stage transforms the input in a form more suitable for the compressor, in order to exploit the known structure.

Another example of compression gain obtained by preprocessing input data can be found in [17], where the input data is constituted of log files of mail servers. A dictionary-based

word replacement phase is performed before applying different lossless compression algorithms and the compression gain is evaluated. Improvements of up to 56 percent in compression time and up to 32 percent in compression ratio are reported. When input data has know structure, an ad-hoc preprocessing can be done: for URL storage and retrieval, in [18] a compression scheme is proposed based on *AVL trees*, a balanced binary tree algorithm that is reported [19], [20] to perform better than red-black trees for lookup-intensive applications. The paper is focused on URL compression and retrieval algorithms for web caches, search engines ad web crawlers. The algorithm is applied to a database of $1.3 * 10^6$ unique URLs obtaining, with average URL length of 55bytes, a reduction in space occupancy of 50% (both store and retrieval online) or 64% (just retrieval online, the case of search engines).

When on the one hand the huge amount of data to be stored and processed is an issue, and on the other hand, some loss of information is acceptable, a trade-off can be applied using lossy compression schemes. In [21] a multi-scale approach is adopted in a system, called *Streaming Warehouse System*, able to process in realtime huge amounts of data, store them, and perform queries on saved data in a reasonable time. A notable characteristic of the presented system is the ability to reply to statistical queries (trend, histograms and correlation) by accessing the compressed version of data, with no need of an intermediate decompression stage. The system implements a multi-scale approach dividing the input time-series in 3 components, with different time and frequency characteristics; each is separately represented with approximations that introduce error (low-pass filtering and sampling, below-threshold clipping, Johnson-Lindenstrauss compressive random projection), achieving data reduction larger than 91%. They also demonstrate that histograms and correlations can be approximated by using the "compressed" data (this is done with known error bounds under some assumptions on the input signal). The paper does not address multi-dimensional data and does not consider correlations among different dimensions.

It is worth noting that all the cited works but [21], [13] either propose a compression algorithm/scheme or a preprocessing stage aimed at improving the compression of the original data: in both categories they gain space efficiency but still have to decompress the data in order to perform any computation on them. In the case of [15], [17], a comparison is possible with our experimental results for the space occupancy, as the input data format they consider can be used in principle also for our input data, and they both refer to plain *bzip2* outcome as we do. The other works only refer to the original size, and consider data types that constitute a subset of ours, therefore a quantitative comparison of the results is not directly possible.

VIII. Conclusion

With the growth of the telecommunication networks and of their user base, the need for efficiently collecting, transferring, processing and storing network monitoring data continuously poses new challenges. We presented a technique that stores high-volume network monitoring data in a representation format that satisfies two main objectives at the same time: the efficient utilization of storage space, and the possibility to perform a number of operations in a computationally convenient way and directly on the transformed data. These properties are traded-off with a controlled loss of accuracy. We explained in detail the different phases composing out technique and the various input parameters, which allow to fine-tune the achievable performance according to different kinds of applications. We also conducted a in-deep evaluation of the technique using data from two different operational networks. The experimental evaluation showed that a compression ratio down to about 20 percent of the original size can be achieved with relative error in the order of 5 percent of the true value (for the numerical fields). With a more aggressive approximation, compression levels larger than that of *bzip2* are reached while preserving the property of being directly searchable and processable. In this case, the accuracy of some non-numerical fields may become quite low. This operating conditions may be used by applications not requiring exact decoding of the original values of these fields. An example is the detection of classes of behavior, e.g. identifying different categories of usage for a number of users (IDs) or a number of services (URLs), which can be used by an operator for QoS/QoE management (using a quasi-static view) or for anomaly detection (using a dynamic view), as reported in Sec. III. The analysis of the sensitivity and scalability also provided interesting results. Properly using a weighting vector allows to trade-off the accuracy on different fields of the input data, accommodating the requirements of different applications. The technique shows also interesting scaling properties with sizes of the input data spanning three orders of magnitude: we observed that, especially for numerical fields, the performance are slightly affected when the size of the input log ranges from 500 to $\sim 100,000$ sessions.

The experiments showed also space for further improvements: an optimized choice of the precision of the matrix elements (in terms of amount of bits for each field) would easily improve the compression ratio. Our ongoing work is focused on researching optimizations reducing the memory footprint of CCS representation by lowering the precision of values in the C matrix from 32 to 16 bit.

Current work is also focused on evaluating the trade-off in accuracy vs compression in relation to different precisions of matrix element representation, and on evaluating the accuracy obtainable after the data-mining operations, in relation to different use cases. Finally, we extend and apply the proposed technique to monitoring data from traffic analysis and characterization [22] and quality of service parameters [23].

Acknowledgment

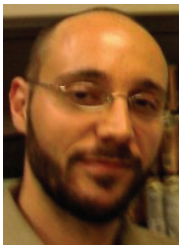
The research has been partially funded by PLATINO (PON01 01007), by MIUR and by "Un sistema elettronico di elaborazione in tempo reale per lestrazione di informazioni da video ad alta risoluzione, alto frame rate e basso rapporto segnale rumore" project of the F.A.R.O. programme. Part of the research has been performed during a summer internship at Docomo Innovation in Palo Alto, CA by Mr. G. Aceto.

References

[1] G. Aceto, A. Botta, A. Pescapé, and C. Westphal, "An efficient storage technique for network monitoring data," in *2011 IEEE International Workshop on Measurements & Networking*.

[2] S. Agrawal, C. N. Kanthi, K. V. M. Naidu, J. Ramamirtham, R. Rastogi, S. Satkin, and A. Srinivasan, "Monitoring infrastructure for converged networks and services," *Bell Labs Technical J.*, vol. 12, no. 2, pp. 63–77, 2007.

[3] M. Peuhkuri, "A method to compress and anonymize packet traces," in *Proc. 2001 ACM Internet Measurement Workshop*, pp. 257–261.
[4] P. B. Ros, G. Iannaccone, J. S. Cuxart, D. A. López, and J. S. Pareta, "Load shedding in network monitoring applications," in *Proc. 2007 USENIX Annual Technical Conference*.
[5] M. Munk, J. Kapusta, and P. Svec, "Data preprocessing evaluation for web log mining: reconstruction of activities of a web visitor," *Procedia Computer Science*, vol. 1, no. 1, pp. 2273–2280, 2010.
[6] B. Claise, G. Sadasivan, V. Valluri, and M. Djernaes, "Cisco systems netflow services export version 9," RFC 3954, Oct. 2004, Tech. Rep.
[7] J. Zujovic and O. G. Guleryuz, "Complexity regularized pattern matching," in *Proc. 2009 IEEE International Conference on Image Processing*.
[8] M. Aharon, M. Elad, and A. M. Bruckstein, "K-SVD and its non-negative variant for dictionary design," *Wavelets XI*, vol. 5914, no. 1, 2005.
[9] K. Cho, K. Mitsuya, and A. Kato, "Traffic data repository at the WIDE project," in *Proc. 2000 USENIX Annual Technical Conference*.
[10] Y. Saad, "SPARSKIT: a basic tool kit for sparse matrix computations," Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA, Tech. Rep. RIACS-90-20, 1990.
[11] Y. Liu, D. Towsley, T. Ye, and J. Bolot, "An information-theoretic approach to network monitoring and measurement," in *2005 IMC*.
[12] A. Pescapé, "Entropy-based reduction of traffic data," *IEEE Commun. Lett.*, vol. 11, no. 2, pp. 191–193, Feb. 2007.
[13] J. Shieh and E. J. Keogh, "iSAX: indexing and mining terabyte sized time series," in *KDD*, Y. Li, B. Liu, and S. Sarawagi, editors. ACM, 2008, pp. 623–631.
[14] S. Vucetic, "A fast algorithm for lossless compression of data tables by reordering," *2006 Data Compression Conference*, vol. 0, pp. 469+.
[15] P. Skibiński and J. Swacha, "Fast and efficient log file compression," in *Proc. CEUR Workshop, 2007 East-European Conference on Advances in Databases and Information Systems*.
[16] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, May 1977.
[17] F. Otten, B. Irwin, and H. Thinyane, "Evaluating text preprocessing to improve compression on maillogs," in *Proc. 2009 ACM SAICSIT*, pp. 44–53.
[18] K. K. Arsa and S. Sanguanpong, "In-memory URL compression," in *Proc. 2001 National Computer Science and Engineering Conference*, pp. 425–428.
[19] B. Pfaff, "Performance analysis of BSTs in system software," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1, pp. 410–411, 2004.
[20] M. AdelsonVelskii and E. Landis, *An Algorithm for the Organization of Information*. Defense Technical Information Center, 1963.
[21] G. Reeves, J. Liu, S. Nath, and F. Zhao, "Managing massive time series streams with multi-scale compressed trickles," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 97–108, 2009.
[22] A. Dainotti, A. Pescapè, and G. Ventre, "A packet-level characterization of network traffic," in *Proc. 2006 CAMAD*, pp. 38–45.
[23] R. Karrer, I. Matyasovszki, A. Botta, and A. Pescapè, "Experimental evaluation and characterization of the *magnets* wireless backbone," in *Proc. 2006 WINTECH*, pp. 26–33.



Giuseppe Aceto is a third year Ph.D. student in Electronic and Telecommunications Engineering at the Department of Biomedic, Electronic and Telecommunication Engineering of University of Napoli Federico II, where he received his MS degree in Telecommunication Engineering in 2008, defending a thesis about a unified platform for available bandwidth estimation in heterogeneous IP networks. Giuseppe Aceto is working in the field of Network monitoring and measurement with focus on internet outages and censorship.

Alessio Botta is a postdoc at the Department of Computer Engineering and Systems of the University of Napoli Federico II (Italy). He graduated in Telecommunications Engineering (M.S.) and obtained the Ph.D. in Computer Engineering and Systems, both at University of Napoli Federico II. His research interests are in the area of networking and, in particular, in the area of network performance measurement and improvement, with a specic focus on wireless and heterogeneous systems. Alessio Botta has coauthored more than 40 international journal (*IEEE Communications Magazine*, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, *Elsevier Computer Networks*, etc.) and conference (IEEE Globecom, IEEE ICC, IEEE ISCC, etc.) publications. He has served and serves several technical program committees of several international conferences (IEEE Globecom, IEEE ICC, etc.) and he acts as reviewer for different international conferences (IEEE Infocom, etc.) and journals (IEEE TRANSACTIONS ON MOBILE COMPUTING, *IEEE Network*, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, etc.) in the area of networking. In 2010 he was awarded with the best local paper award at IEEE ISCC 2010. Alessio Botta has served and serves as independent reviewer of research and implementation project proposals for the Romanian government.

Antonio Pescapé is an Assistant Professor at the Department of Computer Engineering and Systems of the University of Napoli Federico II (Italy) and Honorary Visiting Senior Research Fellow at the School of Computing, Informatics and Media of the University of Bradford (UK). He received the M.S. Laurea Degree in Computer Engineering and the Ph.D. in Computer Engineering and Systems, both at University of Napoli Federico II. Antonio Pescapé teaches courses in Computer Networks, Computer Architectures, Programming, and Multimedia and he has also supervised and graduated more than 100 among BS, MS, and Ph.D. students. His research interests are in the networking field with focus on Internet Monitoring, Measurements and Management and on Network Security. Antonio Pescapé has co-authored over 130 journal (*Communications of the ACM*, *IEEE Communications Magazine*, JSAC, *IEEE Wireless Communications Magazine*, *IEEE Network*, etc.) and conference (SIGCOMM, IMC, PAM, Globecom, ICC, etc.) publications and he is co-author of several patents pending. He has served and serves as workshops and conferences Chair and on more than 90 technical program committees of IEEE and ACM conferences. He has served as Editorial Board Member of IEEE SURVEYS AND TUTORIALS (2008-2011) and was guest editor for the special issue of *Computer Networks* on "Traffic classification and its applications to modern networks". For his research activities he has received several awards. He is a Senior Member of the IEEE. Antonio Pescapé has served and serves as independent reviewer/evaluator of research and implementation projects and project proposals co-funded by the Sweden government, several Italian local governments, Italian Ministry for University and Research (MIUR) and Italian Ministry of Economic Development (MISE).

Cedric Westphal is a Principal Research Architect with Huawei Innovations working on future network architectures, both for wired and wireless networks. His current focus is on Information Centric Networks. He also has been an adjunct assistant professor with the University of California, Santa Cruz since 2009. Prior to Huawei, he was with DOCOMO Innovations from 2007-2011 in the Networking Architecture Group. His work at DOCOMO has covered several topics, all related to next generation network architectures: scalable routing, network virtualization and reliability, using social networks for traffic offloading, etc. Prior to that, he was at Nokia Research Center from 2000 to 2006. He received a MSEE in 1995 from Ecole Centrale Paris, and a MS (1995) and Ph.D. (2000) in EE from the University of California, Los Angeles. Cedric Westphal has co-authored over fifty journal and conference papers, including several best paper awards; and been awarded twenty patents. He has been an area editor for the ACM/IEEE TRANSACTIONS ON NETWORKING since 2009, an assistant editor for (Elsevier) *Computer Networks* journal, and a guest editor for *Ad Hoc Networks* journal. He has served as a reviewer for the NSF, GENI, the EU FP7, and other funding agencies; he has co-chaired the program committee of several conferences, including IEEE ICC (NGN symposium). He is a senior member of the IEEE.