On-demand Time-decaying Bloom Filters for Telemarketer Detection

Giuseppe Bianchi
CNIT / Univ. Roma Tor Vergata
giuseppe.bianchi@uniroma2.it

Nico d'Heureuse, Saverio Niccolini
NEC Laboratories Europe, NEC Europe Ltd
{dheureuse,niccolini}@neclab.eu

ABSTRACT

Several traffic monitoring applications may benefit from the availability of efficient mechanisms for approximately tracking smoothed time averages rather than raw counts. This paper provides two contributions in this direction. First, our analysis of Time-decaying Bloom filters, formerly proposed data structures devised to perform approximate Exponentially Weighted Moving Averages on streaming data, reveals two major shortcomings: biased estimation when measurements are read in arbitrary time instants, and slow operation resulting from the need to periodically update all the filter's counters at once. We thus propose a new construction, called On-demand Time-decaying Bloom filter, which relies on a continuous-time operation to overcome the accuracy/performance limitations of the original window-based approach. Second, we show how this new technique can be exploited in the design of high performance stream-based monitoring applications, by developing VoIPSTREAM, a proof-of-concept real-time analysis version of a formerly proposed system for telemarketing call detection. Our validation results, carried out over real telephony data, show how VoIPSTREAM closely mimics the feature extraction process and traffic analysis techniques implemented in the offline system, at a significantly higher processing speed, and without requiring any storage of per-user call detail records.

Categories and Subject Descriptors

C.2.3 [**Computer Communication Networks**]: Network Operations—*Network Monitoring*

General Terms

Algorithms, Design, Performance, Measurement

Keywords

Rate metering, Bloom filters, monitoring, VoIP, spam, telemarketing detection

1. INTRODUCTION

Several traffic monitoring applications require the ability to discriminate very few specific flows among a large aggregate, on the basis of features or characteristics extracted from the analysis of their traffic patterns.

Whenever the per-flow metrics to be tracked may be expressed in terms of *counts*, efficient and scalable approximate primitives not requiring to keep explicit per-flow states are available, starting from the seminal works of Estan and Varghese (Multi-stage filters [1]), Cohen and Matias (Spectral Bloom filters [2]), and Cormode and Muthukrishnan (Count-Min sketches [3]).

However, several real world applications do not directly handle raw counts (of bytes, packets, durations, events, or combinations of) but require by-products such as smoothed running time averages. For instance, the proof-of-concept use-case application considered in this paper, the VoIP SEAL telemarketing call detection system [4], requires to track, on a *per-call basis*, per-user measurements smoothed over *very long* time periods (several hours). More specifically, VoIP SEAL stores all Call Detail Records (CDRs) into a database, and then derives per-caller metrics by processing the stored CDRs using moving averages over sliding time windows. This operation brings about at least two practical concerns. First, its reliance on database queries makes it hard to adapt it to an online, real-time, operation. Second, the need to store CDRs for a specific analysis purpose poses supplementary deployment hurdles. Indeed, in many countries, data protection laws set forth tight requirements with regard to the storage of telephony data, and may explicitly impose (e.g., in Italy) severe measures on the the security of the stored data and the relevant access control policies.

Obviously, performance, scalability, and storage concerns are best addressed by developing monitoring applications directly able to operate on streaming data. However, the stream-based re-design of a realistic monitoring application such as VoIP SEAL requires to address two main challenges. The first is application-specific, and consists in finding means to compute possibly non trivial metrics (such as per-caller rate of outbound calls addressed towards *new* callees) in a purely online manner. The second, more basic one, consists in devising techniques somewhat similar to the approximate counting approaches discussed above, but specifically designed to track *running averages* rather than counts. Indeed, the direct solution of using approximate counting techniques over non overlapping time windows, and then smoothing the results via a moving average, appears unpractical. A too large window would yield delayed results (being results representative only at the end of the window), and would prevent from detecting rate changes on a time scale smaller than the window itself; a too small window would require to store a huge amount of count sketches to face smoothing time scales in the order of several hours, as in our case.

Our Contribution

This work brings about two major complementary contributions. First, we analyze a formerly proposed data structure, called Time-decaying Bloom filter [5], devised to implement an Exponentially Weighted Moving Average (EWMA) filtering of (rate) measurements. We show that its window-based operation yields two important shortcomings in terms of both accuracy (results being biased when sampled at arbitrary instants of time) and performance (suffering from the need to periodically update all the filter's counters at once). To overcome such limitations, we then propose a novel (to the best of our knowledge) continuous-time construction which we call *On-demand Time-decaying Bloom filter*. It revolves on the suitable integration, in a sketch structure, of a filtering technique referred to as Timed Exponentially Weighted Moving Average [6], which we derive as the limiting case of an ordinary EWMA construction.

Our second contribution consists in exploiting such new construction for designing VoIPSTREAM, a stream-based implementation of the VoIP SEAL monitoring system [4] devised to assist telecom operators in detecting telemarketing spam. VoIPSTREAM's design permits us to show that a non trivial offline monitoring application, leveraging extensive data collection and storage, may be turned into an online application not requiring any storage of data, but capable to analize traffic data on the fly and able to boost performance of a significant factor. In this design, we take a *proof-of-concept* approach. On purpose, our goal is *not* to improve (at this stage) the detection accuracy of the original VoIP SEAL system, but, rather, to show that its operation can be accurately mimicked by the stream-based VoIPSTREAM approach. We believe that our work may provide guidelines and hints to monitoring applications' developers for a similar more performing and storage-free redesign of their own applications.

2. BACKGROUND

A **Bloom filter** [7, 8] is a compact probabilistic data structure used to represent set memberships. In Bloom filters, false positives are possible but false negatives are not. A Bloom filter is implemented as a bit array $b[1:m]$ of m bits accessed via k hash functions $H_1(x) \cdots H_k(x)$, each of which maps a set member x to one of the m bits within the bit array. A Bloom filter BF supports two primitives:

- *BF.Insert(x)*. Insertion of a set member x consists of setting $\forall i \in \{1 \cdots k\}, b[H_i(x)] \leftarrow 1$.

- *BF.Check(x)*. Querying the presence a set member x consists of computing $\min_{i\in\{1\cdots k\}} b[H_i(x)]$, i.e., returning 1 only if *all* corresponding bits are 1.

A **counting Bloom filter** is a generalization of a Bloom filter. Instead of using a bit array, it relies on a *bin* array $B[1:m]$ of m cells. Each bin contains s bits and hence can assume a value in the range $0 \cdots 2^s - 1$.

Originally introduced in [9] as Bloom filters which further permit set member deletion, Counting Bloom filters can be used to approximately count the number of occurrences or account for quantities such as generated bytes for each element within the set. The count filters/sketches employed in [1, 2, 3], even if differing in names and in some technical or implementation details (counters' organization, hash functions employed, their digest size, specific optimizations, etc), may be roughly considered to be variations or extensions of the basic Counting Bloom Filter here reviewed.

For "pure" counting applications (i.e. no per-element decrement required), the counting accuracy can be significantly improved (asymptotically up to a factor k [2]) by exploiting the so called **Conservative Update** [1] or **Minimum Increase** [2] element addition rule. In this paper, we exploit such optimization; accordingly, we leverage the following two primitives of a Counting Bloom Filter CBF:

- *CBF.Count(x)*. Querying an accumulated quantity (e.g., number of occurrences) stored in the filter for a set member x consists of computing the minimum value $\min_{i\in\{1\cdots k\}} B[H_i(x)]$ among the relevant bins.

- *CBF.Add(x,q)*. With the conservative update rule, addition of a quantity $q > 0$ to a set member x consists of determining the minimum value $Count(x)$ among the bins corresponding to the set element x, and then incrementing up to $(Count(x) + q)$ *only* the bins currently lower than this value, i.e., $\forall i \in \{1 \cdots k\}$, $B[H_i(x)] \leftarrow \max\{B[H_i(x)], \min\{Count(x)+q, 2^s-1\}\}$.

Finally, we recall that a Counting Bloom Filter does not strictly require bins to be integer values, although they are typically meant (and developed) as such. Support for **fixed point arithmetic** is straightforward, as it simply requires to conventionally *interpret* a given number of least significant bits in the deployed bins as decimal part, and consistently adapt the input to such a convention (no multiplications are required in the *Add(x,q)* and *Count(x)* primitives). The deployment of **floating point** bins may be convenient in non performance/memory critical software-based implementations (e.g. by using the standard 32 bit float or 64 bit double C/C++ types), when multiplications or exponentiations over bins are required. This is indeed the case of the data structures discussed below.

3. ON-DEMAND TIME-DECAYING BLOOM FILTER

In this section, we first revisit the Time-decaying Bloom Filter (TBF) construction introduced in [5]. Its window-based operation exhibits two notable shortcomings: i) a bias occurring when results are read at *arbitrary* time instants, and, perhaps more crucial in our application' scenario, ii) severe performance limitations posed by the need to periodically decrement all bins at once. These issues motivate the new approach presented in section 3.2.

3.1 Window-based TBF

For sake of clarity we present such construction independently of [5]. Specifically, we first derive it for a single (exact, per-flow) counter case, we then extend it to a Bloom-type filter construction, and we finally discuss the emerging issues and limitations.

Single counter case

Assume, non restrictively, we wish to meter the rate of arrival of calls. Let $C(t)$ be an ordinary call counter, incremented by 1 at each new call arrival. Let W be a (typically short) *measurement window*, and let $0 < \alpha < 1$ be a *Smoothing Coefficient*. At the end of each measurement window, let us update the counter $C(t)$ as follows:

$$C(t)|_{t=nW^+} \leftarrow \alpha \cdot C(t)|_{t=nW^-} \qquad (1)$$

where nW^+ and nW^- are the instants of time right before and right after the end of the nth monitoring window.

To show that this simple rule yields an (implicit) EWMA rate meter, we conveniently define an associated *discrete time* counter $D_n = C(t)|_{t=nW^+}$, obtained by sampling the counter $C(t)$ *right after* the end of a measurement window. Being A_{n-1} the random variable summarizing the number of calls arrived during the measurement window $n-1$, the discrete time counter D_n can be recursively expressed as

$$D_n = \alpha \cdot (D_{n-1} + A_{n-1}) \tag{2}$$

It is easy to show that the call arrival rate R_n (calls/second) is related to D_n by the expression:

$$R_n = D_n \frac{1-\alpha}{\alpha W} \tag{3}$$

Indeed, it suffices to show that, for a stationary arrival process, the expected value of the discrete time random process R_n equals the average call arrival rate. In stationary conditions, $\forall n$, $E[A_{n-1}] = \bar{A}$ is constant. Similarly, $E[D_n] = E[D_{n-1}] = \bar{D}$. By taking expectation on both sides of equation (2), we obtain:

$$E[D_n] = \alpha \left(E[D_{n-1}] + E[A_{n-1}]\right) \to \bar{D} = \bar{A}\frac{\alpha}{1-\alpha} \tag{4}$$

We thus conclude, from (3) and (4), that, for all n,

$$E[R_n] = \bar{D}\frac{1-\alpha}{\alpha W} = \frac{\bar{A}}{W}$$

where we remark that $\bar{A}/W$, the ratio between the average number of calls in a window and the window size W, defines the average call arrival rate.

An explicit recursive expression for R_n can be also provided. By inverting equation (3), and substituting it into (2), after algebraic simplification we obtain

$$R_n = \alpha R_{n-1} + (1-\alpha)\frac{A_{n-1}}{W} \tag{5}$$

This equation confirms that the rate R_n obtained through (3) is an ordinary EWMA-smoothed rate measurement using the parameter α and a window step W.

Window-based TBF Construction

It is straightforward to generalize the approach presented above to the multiple counters' Counting Bloom filter's case, thus obtaining the Time-decaying Bloom filter construction introduced in [5]. Addition of a new quantity is performed exactly as in the case of a traditional CBF, whereas *all* the CBF bins are periodically reduced by a factor α at the end of each monitoring window W.

Such an extension is made possible by noting that a periodic decrement operation, when *simultaneously* performed on all the CBF bins, does not add any further approximation to the CBF operation. Indeed, let us focus on the instants of time $t^- = nW^-$ and $t^+ = nW^+$ right before and right after the end of a monitoring window. Let $B[1:m]$ be the CBF array of m bins, and let us introduce a subscript notation, where $Count_{t^*}(.)$ denotes a query performed at time t^*, whereas $B_{t^*}[.]$ denotes the value of a bin at time t^*. Since, by construction, the multiplication by a *same* factor α is performed for all bins at once, irrespective of the chosen element x the following equality trivially holds:

$$\begin{aligned} Count_{t^+}(x) &= \min_{i\in\{1\cdots k\}} B_{t^+}[H_i(x)] = \min_{i\in\{1\cdots k\}} \alpha \cdot B_{t^-}[H_i(x)] \\ &= \alpha \cdot \min_{i\in\{1\cdots k\}} B_{t^-}[H_i(x)] = \alpha \cdot Count_{t^-}(x) \end{aligned}$$

Discussion and Limitations

The first limitation of the window-based Time-decaying Bloom filter concerns a **bias in the rate estimation**. Indeed, the above treatment relies on the fact that R_n is expressed in terms of the state D_n of the counter (or bins) *when sampled right after the end of the monitoring time window*. It is immediate to see that the relation (3) between the smoothed rate and the counter value does not hold in the general case of counter sampled *during* the window. For instance, if we repeat the above derivation using, instead of D_n, an associated discrete time counter $D'_n = C(t)|_{t=nW^-}$ obtained by sampling the counter $C(t)$ right *before* the end of a measurement window, we would obtain the different relation $R'_n = D'_n(1-\alpha)/W$. We thus conclude that the Window-based TBF, if randomly sampled, is affected by an error inversely proportional to the smoothing coefficient α; for instance, with $\alpha = 0.5$ such error can be up to 100%, whereas $\alpha = 0.95$ approximately yield an up to 5% error.

In the practical cases in which this bias is considered not acceptable, two possible approaches can be envisioned to restore an unbiased rate estimate, but both approaches bring about practical inconveniences. A first one consists in *enabling* the reading of the sketch only at the very first event arriving in the measurement window. This can be accomplished by associating a separate Bloom Filter, reset at each window start time, whose goal is to detect and signal the first event (per each member set) inserted in the sketch. A second approach consists in *directly* constructing the EWMA filter (5) by using two sketches: one for storing the past smoothed measurement, another for collecting the new arrivals in the measurement window time. In both cases, the price to pay is either extra memory, as well as delayed response, as the filter's reading refers to the end of the previous window.

Irrespective of the above discussion, the second, and most notable, concern emerged with our initial experiments with the window-based TBF was its **severe performance limitations**. Indeed, the decrement of all bins at once becomes a critical issue when large filters are deployed. Note that this problem is not unique for SW implementations, but it may easily emerge also in HW implementations, when bins are deployed in Block RAM, being this a very convenient implementation choice over FPGAs in terms of space consumption (see the somewhat related discussion in [10]).

These considerations motivated us to introduce the new TBF construction described in the next section.

3.2 Continuous-time TBF

Periodic update of all the filter's bins is the performance-limiting operation, being linear with respect to the number m of deployed bins. The construction presented in this section circumvents such periodic update by employing an *on-demand* update approach, which consists in *simultaneously decaying and updating* only the actual $k << m$ bins involved in each insertion. This is accomplished by deploying *extra memory, devised to time-stamp each individual bin's update.*

Single counter case

Again, let us first focus on the single counter case. A necessary first step is to extend the EWMA filtering from the discrete step-based operation provided by (5), to the continuous time case. This can be accomplished by taking the limit for $W \to 0$ in (5). The final result can be shown to be equivalent to the Time Exponential Weighted Moving Average continuous-time smoothing technique introduced in [6].

In details, let $\beta < 1$ be the *normalized* smoothing coefficient, i.e., the measured rate decay in the unit time (the actual smoothing coefficient in an arbitrary time interval W being thus β^W). Let us focus on a time step $t \to t+W$, and let us assume that exactly one arrival, in most generality with "weight" X_t, occurs in this interval. With this new notation, expression (5) can be reformulated as:

$$R(t+W) - \beta^W R(t) = (1-\beta^W)\frac{X_t}{W} \tag{6}$$

Taking the limit for $W \to 0$ on both sides, we conclude that

$$dR(t) = \log(1/\beta)X_t$$

or, in other words, that upon an arrival of weigth X_t, the measured rate should be updated by simply adding a quantity $\log(1/\beta)X_t$. Moreover, when no arrivals occur, the measured rate decays exponentially with rate β. Thus, we can finally write a recurrent expression which defines the updating rule between two subsequent arrival times:

$$R(t_{k+1}) = R(t_k)\beta^{t_{k+1}-t_k} + \log(1/\beta)X_{k+1} \tag{7}$$

Continuous-time TBF Construction

Unlike (5), the rate updating rule derived in (7) does not require to introduce a discrete time scale common to all filter bins, but requires the filter to "remember", for each bin, the last update and the relevant time elapsed.

Let $B[1\!:\!m]$ be the bin array of size m. We deploy a companion array $T[1\!:\!m]$ whose role is to "time-stamp" the last instant of time in which the corresponding bin has been updated. The On-demand Time-decaying Bloom filter is thus ultimately specified by means of the following procedures.

- *Count(t, x)*. Querying an accumulated quantity for an element x consists of the following steps:
  - independently update all the bins involved in the look-up for the element x to the actual time t: $\forall i \in \{1 \cdots k\}$, $B[H_i(x)] \leftarrow B[H_i(x)] \cdot \beta^{t-T[H_i(x)]}$;
  - update the relevant bins' age (not strictly necessary, but conveniently performed at this stage): $\forall i \in \{1 \cdots k\}$, $T[H_i(x)] \leftarrow t$;
  - finally return the actualized count, computed as the minimum value among the corresponding bins: $\min_{i\in\{1\cdots k\}} B[H_i(x)]$.
- *Add(t, x,q)*. Addition at time t of a quantity $q > 0$ to a set member x consists of determining the minimum actualized value $Count(x)$ among the bins corresponding to the set element x, computing the updating quantity $q^* = q \cdot \log(1/\beta)$, and then, following the usual conservative update rule, increment up to $(Count(x) + q^*)$ *only* the bins currently lower than this value, i.e., $\forall i \in \{1 \cdots k\}$, $B[H_i(x)] \leftarrow \max\{B[H_i(x)], \min\{Count(x)+q^*, 2^s-1\}\}$.

4. VOIPSTREAM DESIGN

The previously introduced smoothing techniques, combined with recent advances in Bloom-based filtering techniques (e.g., the variation detection approach proposed in [11]), provide a powerful set of tools for the stream analysis (re)design of realistic offline monitoring applications. The proof-of-concept use case considered in this paper is VoIP SEAL (Voice over IP SEcure Application Level firewall). This application, detailed in [4], is meant to assist operators in the process of detecting telemarketing spam. VoIP SEAL classifies users according to a combination of specific metric values, derived from call-level statistics. Features which often characterize telemarketers include a large number of outgoing calls to a large number of distinct users, a small number of incoming calls, a short average call duration, etc.

VoIP SEAL operates in an offline mode. It imports a set of call detail records (CDRs) into a database which is then used for data analysis. Although this implementation reaches an analysis speed of – depending on the data – 300-1000 calls per second, this is not sufficient for an online or near-real-time analysis. Moreover, the need to store CDRs before analyzing them may impair deployment, as CDRs are personal data subject to tight security concerns and regulatory provisions.

Our proposed VoIPSTREAM's streaming operation addresses both these two shortcomings, by providing a dramatic performance increase and by operating "on-the-fly" on CDRs, i.e., not requiring their storage in a database. Note that VoIPSTREAM is on purpose *not* designed to improve the VoIP SEAL telemarketing detection ability but aims at mimicking as closely as possible the original VoIP SEAL operation for proving the viability and appealness of stream-based filtering techniques.

4.1 VoIP SEAL: Input Data and Heuristics

Each CDR analyzed by VoIP SEAL contains the source and destination phone number or identifier, the time the call started, the call duration, and the response code (i.e., SIP reply codes [12]) or cause code (i.e., ITU Q.850 cause codes [13]) which specify whether an error occurred during the call or if the call was successfully established.

For each "user", identified in terms of the caller number and considered of interest only if it has generated at least 30 calls in a 6 or 24 hours sliding window, VoIP SEAL computes *per each incoming call* the following features, named Risk Assessment Modules (RAMs):

- **FoFiR** - Ratio between no. of outgoing (fanout) and incoming (fanin) *established* calls within a 6 hours sliding window (if fanin=0 then FoFiR=fanout);
- **URL** - Ratio between no. of outgoing calls established towards distinct callees (newcallees) and total no. of established calls (fanout) in a 6 hours sliding window;
- **ACD** - Ratio between global average call duration and user's average call duration, both measured in a 24 hours sliding window.

Additionally, a fourth RAM, called SOC and defined as the number of concurrently active outgoing calls per user, is employed in the original VoIP SEAL system. However, experimental results reveal that its influence on the classification

Table 1: Algorithm settings

Alg.	t_1	t_2	w_i	Time window (hours)
FoFiR	2	10	2	6
URL	0.5	1	3	6
ACD	5	10	3	24

is marginal (for the used dataset), and as such not worth the (non trivial) online implementation effort.

Each RAM i analyzes the data independently and assigns a score $s_i(c)$ to each call c. This score is calculated by transforming an algorithm-specific metric $m_i(c)$ into a score between 0 and 1 using a linear interpolation between the two thresholds $t_{1,i}$ and $t_{2,i}$. The RAM metrics are designed such that calls placed by telemarketers receive higher scores. The weighted sum of these individual RAM scores is used as a total call score $s(c)$. The higher the score, the more likely the call is to be initiated by a telemarketer. The RAM parameters and their weights w_i are summarized in Table 1.

4.2 VoIPSTREAM implementation

Unlike VoIP SEAL, VoIPSTREAM operates on the fly on incoming call events (currently read from CDRs), keeping track of the past activity only implicitly, through a number of On-demand Time-decaying Bloom filters (section 3.2) plus a variation detector discussed below.

First, the CDR fields associated to a call are parsed and the following values are extracted: *timestamp*, *caller id*, *callee id*, *call duration*, and *call established* (a boolean flag determined from the call's reply code).

A further flag, hereafter referred to as *new callee*, is set whenever the current caller establishes a call towards a callee not previously contacted within a given time frame (the last 6 hours in the VoIP SEAL offline system). For computing such flag, we employ the *Variation detector* construction introduced in [11]. Such construction is composed of two Bloom filters, a *detecting* filter and a *learning* one. Both filters are updated with the `<caller id,callee id>` pair associated to the incoming call. Additionally, the detecting filter is in charge of setting the *new callee* flag to 1 whenever such pair is not previously stored in the filter itself. When the detecting filter is saturated, the filters are swapped such that the learning filter becomes the detecting filter and conversely. The new learning filter is cleared, so that it begins each cycle empty. The purpose of this arrangement is to ensure that the detecting filter starts each new cycle "warm"; the alternative, a single periodically reset detecting filter, would over-count already-observed pairs flushed on each reset. The size of the filters composing the variation detector roughly determines the time period over which calls are tracked. A 2^{21} bits filter - i.e. only 250 KB - using $k = 8$ hash functions - i.e. less than 0.4% false positive - permits to track the latest 180.000 distinct `<caller,callee>`, enough to mimic the required 6 hours window of traffic on the considered data set.

The core of VoIPSTREAM consists of 5 On-demand Time-decaying Bloom he VoIP SEAL RAMs' arameters, the three filters used to derive the FoFiR and the URL scores employ a smoothing constant $\beta = .9672$ corresponding to a 6 hours filtering memory, whereas the remaining two, used to derive the ACD score, employ $\beta = .9917$ corresponding to a 24 hours filtering memory. The deployed filters are:

- *RCR: per-user received call rate* (smoothing: 6h). Updated if the call is established, by adding the callee to the RCR filter, i.e., *RCR.Add(timestamp, callee, 1)*; permits to read the *fanin* rate for the current user (caller) as *fanin = RCR.Count(timestamp, caller).*
- *ECR: per-user established call rate* (smoothing: 6h). Updated if the call is established, by adding the caller to the ECR filter, i.e., *ECR.Add(timestamp, caller, 1)*; permits to read the *fanout* rate for the current user (caller) as *fanout = ECR.Count(timestamp, caller).*
- *ENCR: per-user new callee rate* (smoothing: 6h). Updated if the call is established and if the *new callee* flag is set, by adding the user (caller) to the ENRC filter, i.e., *ENCR.Add(timestamp, caller, 1)*; permits to read the *calleeRate* for the current user as *calleeRate = ENCR.Count(timestamp, caller).*
- *ECR24: per-user established call rate* (smoothing: 24h). Same as ECR, but with a 24h smoothing constant.
- *CT24: per-user total call time* (smoothing: 24h). Updated if the call is established, by adding the call duration to the CT24 filter, i.e., *CT24.Add(timestamp, caller, duration)*; permits to read the current *calltime* metric, i.e. the total amout of call time generated by the user, as *calltime = CT24.Count(timestamp, caller).*

The above derived metrics are finally used for computing the RAMs for users whose established call generation rate is at least 30 calls per 6 hours. The FoFiR and URL RAMs are directly obtained from the ratios *fanout/fanin* and *calleeRate/fanout* and scored as in the VoIP SEAL case.

The ACD metric requires to first derive the average call duration as the ratio between the per-user *calltime* and the per-user call establish rate (*fanout*), consistently gathered from the ECR24 filter. This value is then compared with the average call duration for all the calls (i.e., the system average call duration), separately accounted via two ordinary (single counter) EWMA filters smoothed with a 24h constant, and respectively tracking the total number of established calls and the total call duration.

Discussion

Time decaying Bloom filters directly inherit from the approximate counting sketches based on the conservative update rule the property that the error introduced by the approximated operation reduces as the metric to be tracked increases. We exploit this property for checking whether the filters are correctly sized for the considered data set, by run-time tracking their accuracy through the periodic invocation of the relevant *Count(.)* primitives on random strings (hence expected to have a count equal to 0, and as such affected by the largest possible error). Over the considered data set, we have experimentally found that filters (conservatively) sized with a number of bins between 2^{19} and 2^{21} (depending on the deployed filter' smoothing constants) exhibit a negligible error.

In our implementation, we have not been challenged by memory concerns. Indeed, even if we use a double precision floating point number for each bin, with the rather conservative filter sizing (discussed above) the total amount of run-time memory allocated by our implementation is approximately 300 MB. This is well withing the capabilities of the host computer employed for the measurements. If memory becomes a concern, several optimization directions appear possible. One possibility is to use techniques analogous to [2, 5] for allocating a different amount of bits to the filter bins. Note, however, that such an optimization alone is less effective in our on-demand construction, as it would not involve the bits allocated to each bins' time-stamp. Another possible optimization consists in trying to operate the filters so that the metrics of interest achieve a larger value (hence a lower error) than those not of interest. This would allow to reduce the number of bins while keeping the error constant. For instance, in the case of the ACD RAM, we are especially interested in tracking short calls which, as such, suffer from greater errors. To reduce the filter sizes used for tracking the ACD RAM, it might thus be better to account, inside the filter, the inverse call duration (*1/callDuration*), and adjust the RAM thresholds accordingly, on the basis of the known or estimated call duration distribution. We leave such memory optimization to future work.

5. EXPERIMENTAL EVALUATION

Thanks to the collaboration with a small European telecom operator, we could run our code over real-world CDRs. The data set consists in more than 30 million CDRs, generated by more than 4 millions of users over a consecutive time period of approximately 5 weeks. Unfortunately, (almost) no ground truth is available for this data set, as only 5 telemarketers were explicitly identified by the operator, and it is thus not possible to reliably assess the "absolute" telemarketer detection accuracy of VoIPSTREAM.

Nevertheless, it is possible to quantitatively assess the ability of VoIPSTREAM to achieve results comparable with that obtained with the offline CDR analysis system VoIP SEAL [4], and thus show to what extent VoIPSTREAM's stream-based operation may approach VoIP SEAL's offline analysis.

We recall that a perfect call-by-call matching is technically unfeasible as the underlying smoothing mechanisms intrinsically differ. VoIPSTREAM relies on (Time) Exponentially Weighted Moving Averages which by construction assign more weight to the most recent events, whereas VoIP SEAL uses sliding windows, hence uniform weights. However, the ultimate goal is to classify whether an user exhibits an anomalous telemarketer-like behavior over the long term, and it seems reasonable to expect (as indeed fully confirmed by the actual results discussed below) that technical differences in the smoothing algorithms employed should not dramatically affect the per-user classification results.

For our experiments, we used a server with an AMD Opteron^{TM}246 CPU (2 cores à 2.0 GHz) and 6GB of memory. Using only one core, VoIPSTREAM achieved a processing performance of approximately 17.000 calls per second[1] (57M calls per hour), already sufficient to handle the complete telephony traffic in most operator network. This is a very promising result, as our SW implementation is still far from being optimized and, particularly, it is still based on a slow cryptographic hash (SHA-1, which performs at least one to two orders of magnitude less than state of the art non cryptographic hashes [14]).

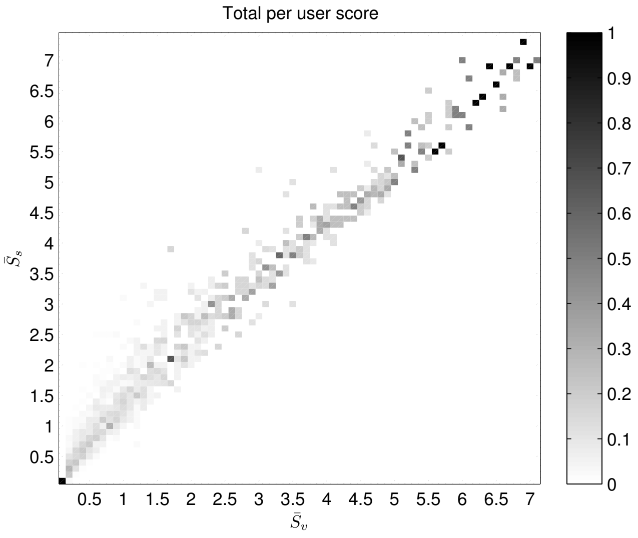

Figure 2: Global score averaged by user

Figures 1 and 2 show how close the stream operation of VoIPSTREAM succeeds in approaching the VoIP SEAL's classification performance. Fig. 1 compares results on a per RAM basis. For each user we calculate, over all scored calls, the average VoIP SEAL and VoIPSTREAM scores per RAM (rounded to one decimal). We then count how many users have a certain VoIP SEAL average score $\bar{S}_v$ while at the same time having a VoIPSTREAM average score of $\bar{S}_s$. This number is then normalized by the total number of users having a VoIP SEAL average score of $\bar{S}_v$. The heatmaps of Fig. 1 shows these normalized values. A perfect matching would yield a diagonal plot, i.e. the value should be *one* if $\bar{S}_v = \bar{S}_s$ and *zero* otherwise. Remarkably, for all the RAMs, we observe an almost perfect diagonal, i.e., a very good match between VoIP SEAL's and VoIPSTREAM's scores. The slight overestimation that can be observed for the VoIPSTREAM scores is most likely imputable to the different type of filtering (EWMA versus sliding window) used by the two systems, rather than different settings. Indeed, we remark that, in order to compare the two systems in the same conditions, the smoothing coefficients β employed in the VoIPSTREAM Time-decaying Bloom filters have been on purpose configured so as to match the time constants of the corresponding VoIP SEAL modules (see section 4.2).

Finally, Fig. 2 shows a normalized comparison of the VoIP SEAL and VoIPSTREAM per-user total average scores, i.e., the weighted sum of the individual RAM scores. Again, we can observe a clear diagonal on the heatmap which corresponds to a good match between the VoIP SEAL and the VoIPSTREAM total scores. For small values of $\bar{S}_v$ there are some discrepancies with the VoIPSTREAM score $\bar{S}_s$, which appear to reduce as the score increases (indeed, our area of interest for user classification purposes).

[1]For performance comparison, we also implemented the VoIPSTREAM system by replacing all the On-demand Time-decaying Bloom filters, with the window-based ones described in section 3.1, and maintaining all the rest of the implementation unvaried (same hash functions, etc). As a result, VoIPSTREAM's performance dropped down to less than 5000 calls/second. Note that this comparison strictly depends on the specific filter size employed; large filters would yield further performance reduction for the window-based construction, whose performance linearly depends on the filter size, whereas it is independent in the on-demand case.

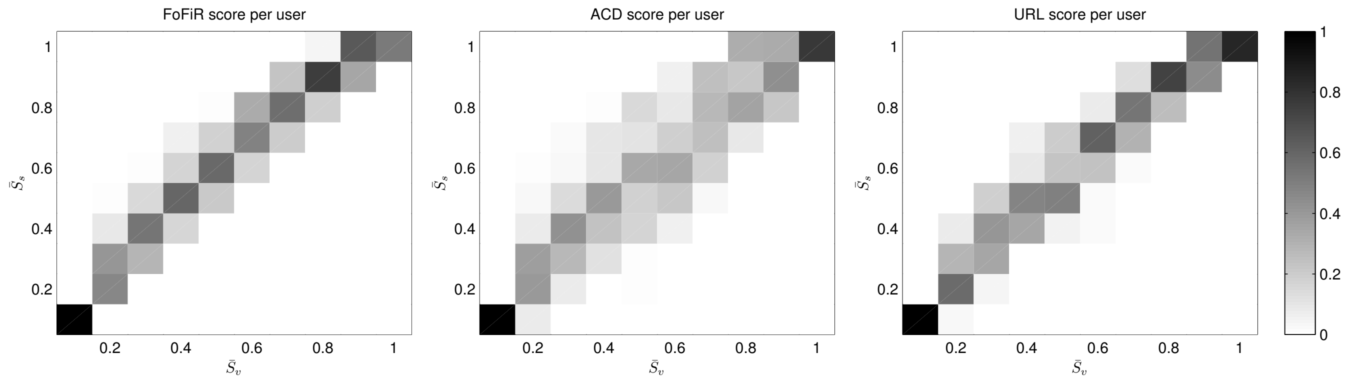

Figure 1: RAM scores averaged by user

6. CONCLUSIONS

In this work we have introduced a new On-demand construction for Time-decaying Bloom filters. Its continuous-time operation permits to improve both accuracy and performance of the original discrete-time Time-decaying Bloom filter operation. We have then exploited such construction for designing VoIPSTREAM, an high performance stream-based version of a formerly proposed system for telemarketing call detection via analysis of telephony logs. Experimental results prove a remarkable ability of VoIPSTREAM in mimicking the classification accuracy of the offline system, but with a significantly higher processing speed, and without requiring any storage of privacy-critical per-user call detail records.

Beyond the punctual technical contribution here documented, we believe that our work, and the methodologies herein documented, may inspire monitoring applications' developers towards a stream-based rethinking of their application's design, thus gaining performance and storage-free operation.

7. ACKNOWLEDGMENTS

This work was partially supported by DEMONS, a research project supported by the European Commission under its 7th Framework Program (contract no. 257315). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the DEMONS project or the European Commission.

8. REFERENCES

[1] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," *SIGCOMM Comput. Commun. Rev.*, vol. 32, pp. 323–336, August 2002.

[2] S. Cohen and Y. Matias, "Spectral bloom filters," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '03. New York, NY, USA: ACM, 2003, pp. 241–252.

[3] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *J. Algorithms*, vol. 55, pp. 58–75, April 2005.

[4] N. d'Heureuse, S. Tartarelli, and S. Niccolini, "Analyzing telemarketer behavior in massive telecom data records," in *21st Tyrrhenian Workshop on Digital Communications: Trustworthy Internet*, 2010.

[5] K. Cheng, L. Xiang, M. Iwaihara, H. Xu, and M. M. Mohania, "Time-decaying bloom filters for data streams with skewed distributions." in *Proc. 15th Int. Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications*, 2005.

[6] R. Martin and M. Menth, "Improving the timeliness of rate measurements." in *12th GI/ITG Conference on measuring, modelling and evaluation of computer and communication systems*, 2004.

[7] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, pp. 422–426, July 1970.

[8] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," in *Internet Mathematics*, 2002, pp. 636–646.

[9] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 254–265, 1998.

[10] S. Pontarelli, S. Teofili, and G. Bianchi, "Hardware-based on-the-fly per-flow scan detector prefilter." in *Third COST TMA International Workshop on Traffic Monitoring and Analysis*, 2011.

[11] G. Bianchi, E. Boschi, S. Teofili, and B. Trammell, "Measurement data reduction through variation rate metering." in *INFOCOM'10*, 2010, pp. 2187–2195.

[12] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261 (Proposed Standard), Jun. 2002, updated by RFCs 3265, 3853, 4320, 4916.

[13] ITU-T, "Usage of cause and location in the Digital Subscriber Signalling System No. 1 and the Signalling System No. 7 ISDN User Part," ITU-T Recommendation Q.850, May 1998.

[14] C. Henke, C. Schmoll, and T. Zseby, "Empirical evaluation of hash functions for packetid generation in sampled multipoint measurements," in *Proceedings of the 10th International Conference on Passive and Active Network Measurement*, 2009.