Sketch-based Change Detection: Methods, Evaluation, and Applications

Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang
AT&T Labs–Research; 180 Park Avenue
Florham Park, NJ, USA
{bala,sen,yzhang}@research.att.com

Yan Chen*
University of California
Berkeley, CA, USA
yanchen@cs.berkeley.edu

ABSTRACT

Traffic anomalies such as failures and attacks are commonplace in today's network, and identifying them rapidly and accurately is critical for large network operators. The detection typically treats the traffic as a collection of flows that need to be examined for significant changes in traffic pattern (*e.g.*, volume, number of connections). However, as link speeds and the number of flows increase, keeping per-flow state is either too expensive or too slow. We propose building compact summaries of the traffic data using the notion of sketches. We have designed a variant of the sketch data structure, *k-ary sketch*, which uses a constant, small amount of memory, and has constant per-record update and reconstruction cost. Its linearity property enables us to summarize traffic at various levels. We then implement a variety of time series forecast models (ARIMA, Holt-Winters, etc.) on top of such summaries and detect significant changes by looking for flows with large forecast errors. We also present heuristics for automatically configuring the model parameters.

Using a large amount of real Internet traffic data from an operational tier-1 ISP, we demonstrate that our sketch-based change detection method is highly accurate, and can be implemented at low computation and memory costs. Our preliminary results are promising and hint at the possibility of using our method as a building block for network anomaly detection and traffic measurement.

Categories and Subject Descriptors

C.2.3 [**Computer-Communications Networks**]: Network Operations—*network monitoring*

General Terms

Measurement, Algorithms

Keywords

Change Detection, Network Anomaly Detection, Data Stream Computation, Sketch, Time Series Analysis, Forecasting

*Work done while at AT&T Labs–Research

1. INTRODUCTION

Traffic anomalies are an integral part of daily life for today's network operators. Some traffic anomalies are expected or unanticipated but tolerable. Others are often indications of performance bottlenecks due to flash crowds [25], network element failures, or malicious activities such as denial of service attacks (DoS) [23] and worms [28]. Suitable motivation exists to process massive data streams (available from diverse sources) quickly to examine them for anomalous behavior.

Two basic approaches to network anomaly detection are common. The first approach is the signature-based approach. It detects traffic anomalies by looking for patterns that match signatures of *known* anomalies. For example, Moore *et al.* [29] infer DoS activities based on address uniformity, a property shared by several popular DoS toolkits. Signature-based methods have been extensively explored in the literature and many software systems and toolkits such as Bro [31] and Snort [32] have been developed and are being used. One limitation of this approach is the requirement that the anomaly signatures be known in advance; thus it cannot be applied to identify new anomalies. Also, a malicious attacker can evade signature-based detection systems by garbling the signatures. One can see a parallel in the failure of filter-based spam fighting systems where spammers introduce random hashes in the spam messages.

The second approach is the statistics-based approach, which does not require prior knowledge about the nature and properties of anomalies and therefore can be effective even for new anomalies or variants of existing anomalies. A very important component of statistics-based approach is *change detection*. It detects traffic anomalies by deriving a model of normal behavior based on the past traffic history and looking for significant changes in short-term behavior (on the order of minutes to hours) that are inconsistent with the model. Our goal in this work is to come up with an efficient, accurate, and scalable change detection mechanism for detecting significant changes in massive data streams with a large number of flows.

1.1 Change detection: existing techniques and limitations

Change detection has been extensively studied in the context of time series forecasting and outlier analysis [35, 36, 12, 13]. The standard techniques include different smoothing techniques (such as exponential smoothing or sliding window averaging), the Box-Jenkins ARIMA modeling [6, 7, 2], and finally the more recent wavelet-based techniques [4, 3].

Prior works have applied these techniques to network fault detection and intrusion detection. Examples in fault detection include [22, 26, 38]. Feather *et al.*identify faults based on statistical deviations from normal traffic behavior [18]; a method of identifying aberrant behavior by applying thresholds in time series models of network traffic is described in [9]. Methods for intrusion detection include neural networks [20], Markov models [40], and clustering [34]. Barford *et al.*recently provide a characterization of different types of anomalies [4] and propose wavelet-based methods for change detection [3].

Unfortunately, existing change detection techniques can typically only handle a relatively small number of time series. While this may suffice for detecting changes in highly aggregated network traffic data (*e.g.*, SNMP link counts with 5 minute sample interval), they cannot scale up to the needs at the network infrastructure (*e.g.*, ISP) level. At an ISP level, traffic anomalies may be buried inside the aggregated traffic, mandating examination of the traffic at a much lower level of aggregation (*e.g.*, IP address level) in order to expose them. Given today's traffic volume and link speeds, the detection method has to be able to handle potentially several millions or more of concurrent network time series. Directly applying existing techniques on a per-flow basis cannot scale up to the needs of such massive data streams. Recent research efforts have been directed towards developing scalable heavy-hitter detection techniques for accounting and anomaly detection purposes [17]. Note that heavy-hitters do not necessarily correspond to flows experiencing significant changes and thus it is not clear how their techniques can be adapted to support change detection.

1.2 Data stream computation and sketches

In the database research community, however, computation over massive data streams has been an active research area over the past several years. The emerging field of *data stream computation* deals with various aspects of computation that can be performed in a space- and time-efficient fashion when each tuple in a data stream can be touched only once (or a small number of times). A good survey of the algorithms and applications in data stream computation can be found in [30]. One particularly powerful technique is *sketch* [24, 21, 15], a probabilistic summary technique proposed for analyzing large streaming datasets. Sketches avoid keeping per-flow state by dimensionality reduction, using projections along random vectors. Sketches have some interesting properties that have proven very useful in data stream computation: they are space efficient, provide provable probabilistic reconstruction accuracy guarantees, and are linear (*i.e.,* sketches can be combined in an arithmetical sense).

1.3 Our contribution

In this work, we incorporate data stream computation techniques into change detection. Our solution, labeled *sketch-based change detection*, is the first one, to the best of our knowledge, that is capable of detecting significant changes in massive data streams with a large number of network time series. With sketch-based change detection, we first build compact summaries of the traffic data using sketches. We have designed a variant of the sketch data structure, *k-ary sketch*, which uses a constant, small amount of memory, and has constant per-record update and reconstruction cost. We then implement a variety of time series forecast models (ARIMA, Holt-Winters, etc.) on top of such summaries and detect significant changes by looking for flows with large forecast errors. Being able to compute significant differences in the list of top flows quickly can point towards *potential* anomalies. Depending on the length of the time period for which we compute forecasts and the duration of significant changes, we can accurately identify the presence of an anomaly. Note that an anomaly can be a benign surge in traffic (like a flash crowd) or an attack. We also present heuristics for configuring the model parameters.

We demonstrate using a large amount of real Internet traffic data that our sketch-based change detection method is highly accurate when compared with per-flow analysis, and can be implemented at low computation and memory costs. Our evaluation shows that we can reconstruct lists of the top flows in a time period efficiently and accurately; we are also able to achieve similar forecast errors when compared with per-flow techniques. While our techniques have not yet been directly applied to anomaly detection, our preliminary results in change detection are promising and we believe that our method can serve as a building block for network anomaly detection.

1.4 Paper outline

The rest of the paper is organized as follows: Section 2 gives an overview of the framework of our sketch-based change detection, followed by detailed discussions of different modules in Section 3. Section 4 describes the experimental setup. Section 5 presents key portions of the results of our extensive testing of sketch-based change detection on different large and real datasets. We summarize our ongoing research in Section 6 and conclude the paper in Section 7.

2. OVERVIEW

2.1 Data stream model

Over the past several years, various models have been proposed to describe data streams, including Time Series Model, Cache Register Model, and Turnstile Model [30]. We use the the most general one—the Turnstile Model.

Specifically, let $\mathcal{I} = \alpha_1, \alpha_2, \cdots$, be an input stream that arrives sequentially, item by item. Each item $\alpha_i = (a_i, u_i)$ consists of a key $a_i \in [u] \stackrel{\text{def}}{=} \{0, 1, \cdots, u-1\}$, and a (possibly negative) update $u_i \in \mathbb{R}$. Associated with each key $a \in [u]$ is a time varying signal $A[a]$. The arrival of each new data item (a_i, u_i) causes the underlying signal $A[a_i]$ to be updated: $A[a_i] += u_i$. The goal of change detection is to identify all those signals with significant changes in their behavior.

The above model is very general and one can instantiate it in many ways with specific definitions of the key and updates. In the context of network anomaly detection, the key can be defined using one or more fields in packet headers such as source and destination IP addresses, source and destination port numbers, protocol number etc. It is also possible to define keys with entities like network prefixes or AS numbers to achieve higher levels of aggregation. The update can be the size of a packet, the total bytes or packets in a flow (when flow-level data is available). To keep the parameter space within a manageable size, however, we only use destination IP address and bytes in the experiments discussed in this paper. Alternative choice of keys and values may affect the running time, but we expect the accuracy results to be quite similar.

2.2 Sketch-based change detection

In an *ideal* environment with infinite resources, we can perform time series forecasting and change detection on a *per-flow* basis. Specifically, we break time into discrete intervals $I_1, I_2, \cdots$. For each time interval I_t, and each signal $A[a]$ that appears before or during interval I_t, we first compute the *observed* value—total update to $A[a]$ during interval I_t: $o_a(t) = \sum_{i \in A_a(t)} u_i$, where the set of indices $A_a(t) \stackrel{\text{def}}{=} \{i \mid a_i = a \wedge (a_i, u_i) \text{ arrives during } I_t\}$. We also compute the *forecast* value $f_a(t)$ by applying a forecasting model to observed values in the past intervals. We then compute the *forecast error* $e_a(t) = o_a(t) - f_a(t)$ and raise an alarm whenever $e_a(t)$ is significant according to certain detection criteria.

In the real world, however, per-flow analysis can be prohibitive because the number of signals present in the input stream can be very large. For instance, if we use source and destination IPv4 addresses as the key, the key space $[u]$ can be as large as 2^{64}, and the number of signals can easily reach tens of millions given today's traffic volume and link speeds. Hence it can be too slow or too expensive to perform change detection on a per-flow basis.

Our solution is to create sketches to summarize the input stream and implement various forecasting models on top of the sketches. Specifically, our *sketch-based change detection* consists of the following three basic modules:

1. Sketch module
2. Forecasting module
3. Change detection module

The first module—sketch module—creates a (space- and time-efficient) sketch to summarize all the observed values $o_a(t)$ (total update to signal $A[a]$) during each time interval I_t—the *observed* sketch $S_o(t)$. The forecasting module produces a *forecast* sketch $S_f(t)$ using some forecasting models based on observed sketches in the past intervals. It then computes the *forecast error* sketch $S_e(t)$ as the delta between $S_o(t)$ and $S_f(t)$, *i.e.*, $S_e(t) = S_o(t) - S_f(t)$. The linearity of the sketch data structure allows us to implement various forecasting models and compute the deltas directly at the sketch level. The change detection module uses the error sketch $S_e(t)$ to determine significant changes. We next describe these modules in details.

3. DETAILS

3.1 Sketch module

Let $(a_1, u_1), (a_2, u_2), \cdots$ be an input stream (for example, the substream of $\mathcal{I}$ that is observed during a given time interval). For each key $a \in [u]$, let $v_a = \sum_{i \in A_a} u_i$, where the set of indices $A_a \stackrel{\text{def}}{=} \{i \mid a_i = a\}$.

For each interval, the *second moment* (F_2) is defined as the sum of squares of the values associated with all the keys, *i.e.*, $F_2 \stackrel{\text{def}}{=} \sum_a v_a^2$. We refer to the square root of the second moment ($\sqrt{F_2}$) as the *L2 norm*.

The sketch module uses the sketch data structure to summarize all the v_a in each time interval. Sketch is a probabilistic summary data structure based on random projections (See [30] for a good overview of sketches and the general field of data stream computation). We have designed a variant of the sketch data structure, which we call the *k-ary sketch*. The *k-ary* sketch is similar to the count sketch data structure recently proposed by Charikar *et al.*[11]. However, the most common operations on *k-ary* sketch use simpler operations and are more efficient than the corresponding operations defined on count sketches [33].

Just like the count sketch, a *k-ary* sketch S consists of a $H \times K$ table of registers: $T_S[i][j]$ ($i \in [H], j \in [K]$). Each row $T_S[i][\cdot]$ ($i \in [H]$) is associated with a hash function from $[u]$ to $[K]$: h_i. We can view the data structure as an array of hash tables. The hash functions are required to be 4-universal [10, 39] to provide probabilistic guarantees of reconstruction accuracy. We construct them using the fast tabulation-based method developed in [33]. Different h_i are constructed using independent seeds, and are therefore independent.

There are four basic operations defined for *k-ary* sketches: UPDATE to update a sketch, ESTIMATE to reconstruct v_a for a given key a, ESTIMATEF2 to estimate the second moment F_2, and COMBINE to compute the linear combination of multiple sketches. They are used in various modules of our change detection scheme: UPDATE in the sketch module to update the observed sketch $S_o(t)$; COMBINE in the forecasting module to implement various forecasting models and to compute the forecast sketch $S_f(t)$ and forecast error sketch $S_e(t)$; ESTIMATE in the change detection module to reconstruct forecast errors from $S_e(t)$; and ESTIMATEF2 in the change detection module to choose the threshold for judging whether forecast errors are significant.

The formal specification of these operations is as follows.

1. UPDATE(S, a, u)**:** For $\forall i \in [H]$, $T_S[i][h_i(a)] += u$.

2. ESTIMATE(S, a)**:** Let $sum(S) = \sum_{j \in [K]} T_S[0][j]$ be the sum of all values in the sketch, which only needs to be computed once before any ESTIMATE(S, a) is called. Return an estimate of v_a

   $$v_a^{\text{est}} = \text{median}_{i \in [H]}\{v_a^{h_i}\}$$

   where

   $$v_a^{h_i} = \frac{T[i][h_i(a)] - sum(S)/K}{1 - 1/K}$$

   As shown in Appendix A, each $v_a^{h_i}$ ($i \in [H]$) is an unbiased estimator of v_a with variance inversely proportional to $(K-1)$. v_a^{est} further improves accuracy by avoiding the extreme estimates.

3. ESTIMATEF2(S)**:** Return an estimate of the second moment

   $$F_2^{\text{est}} = \text{median}_{i \in [H]}\{F_2^{h_i}\}$$

   where

   $$F_2^{h_i} = \frac{K}{K-1} \sum_{j \in [K]} (T_S[i][j])^2 - \frac{1}{K-1}(sum(S))^2$$

   As shown in Appendix B, each $F_2^{h_i}$ forms an unbiased estimator of F_2 with variance inversely proportional to $(K-1)$. F_2^{est} further improves accuracy by avoiding the extreme estimates.

4. COMBINE$(c_1, S_1, \cdots, c_\ell, S_\ell)$**:** The linearity of the sketch data structure allows us to linearly combine multiple sketches $S = \sum_{k=1}^{\ell} c_k \cdot S_k$ by combining every entry in the table:

   $$T_S[i][j] = \sum_{k=1}^{\ell} c_k \cdot T_{S_k}[i][j]$$

3.2 Forecasting module

The forecasting module uses the observed sketches in the past intervals $S_o(t')$ ($t' < t$) to compute the forecast sketch $S_f(t)$ and along with it the error between the observed and forecast sketches as $S_e(t)$. In this work, we explore six models commonly used in univariate time series forecasting and change detection. The first four models are simple smoothing models; the other two belong to the family of ARIMA models. All six models can be implemented on top of sketches by exploiting the linearity property of sketches.

3.2.1 *Simple smoothing models*

The first four models are simple smoothing models and are popular due to their simplicity. They are moving average (MA), exponentially weighted moving average (EWMA), S-shaped moving average (SMA), and non-seasonal Holt-Winters (NSHW).

Moving Average (MA) This forecasting model assigns equal weights to all past samples, and has a single integer parameter $W \geq 1$ which specifies the number of past time intervals used for computing the forecast for time t.

$$S_f(t) = \frac{\sum_{i=1}^{W} S_f(t-i)}{W}, \quad W \geq 1$$

S-shaped Moving Average (SMA) This is a class of weighted moving average models that give higher weights to more recent samples.

$$S_f(t) = \frac{\sum_{i=1}^{W} w_i \cdot S_f(t-i)}{\sum_{i=1}^{W} w_i}, \quad W \geq 1$$

We use a subclass that gives equal weights to the most recent half of the window, and linearly decayed weights for the earlier half (see [19] for discussion).

Exponentially Weighted Moving Average (EWMA) With exponentially weighted moving average, the forecast for time t is the weighted average of the previous forecast and the newly observed sample at time $t-1$.

$$S_f(t) = \begin{cases} \alpha \cdot S_o(t-1) + (1-\alpha) \cdot S_f(t-1), & t > 2 \\ S_o(1), & t = 2 \end{cases}$$

The parameter $\alpha \in [0,1]$ is called the smoothing constant. It indicates how much weight is given to new samples vs. the history.

Non-Seasonal Holt-Winters (NSHW) The Holt-Winters model [8] is another commonly used smoothing model and it has been applied in [9] to detect aberrant behavior. In the non-seasonal Holt-Winters model, there is a separate smoothing component $S_s(t)$ and a trend component $S_t(t)$. There are two parameters $\alpha \in [0,1]$ and $\beta \in [0,1]$.

$$S_s(t) = \begin{cases} \alpha \cdot S_o(t-1) + (1-\alpha) \cdot S_f(t-1), & t > 2 \\ S_o(1), & t = 2 \end{cases}$$

$$S_t(t) = \begin{cases} \beta \cdot (S_s(t) - S_s(t-1)) + (1-\beta) \cdot S_t(t-1), & t > 2 \\ S_o(2) - S_o(1), & t = 2 \end{cases}$$

The forecast is then $S_f(t) = S_s(t) + S_t(t)$.

3.2.2 *ARIMA models*

Box-Jenkins methodology, or AutoRegressive Integrated Moving Average (ARIMA) modeling technique [6, 7], is a class of linear time series forecasting techniques that capture the linear dependency of the future values on the past values. They are able to model a wide spectrum of time series behavior. As a result, they have been extensively studied and widely used for univariate time series forecasting and change detection.

An ARIMA model includes three types of parameters: the autoregressive parameter (p), the number of differencing passes (d), and the moving average parameter (q). In the notation introduced by Box and Jenkins, models are summarized as ARIMA (p, d, q). A model described as $(0, 1, 2)$ means that it contains $p = 0$ (zero) autoregressive parameters and $q = 2$ moving average parameters which were computed for the time series after it was differenced once ($d = 1$). Note that we use only integral values for p, d, and q. Although there has been recent work on models with fractional d (the ARFIMA model) in the context of action-reaction models [27], we have not yet examined their application in the networking context.

A general ARIMA model of order (p, d, q) can be expressed as:

$$Z_t - \sum_{i=1}^{q} MA_i \cdot Z_{t-i} = C + e_t - \sum_{j=1}^{p} AR_j \cdot e_{t-i}$$

where Z_t is obtained by differencing the original time series d times, e_t is the forecast error at time t, MA_i ($i = 1, ..., q$) and AR_j ($j = 1, ..., p$) are MA and AR coefficients.

In practice, p and q very rarely need to be greater than 2. The number of differences (d) is typically either 0 or 1. Therefore, when we extend ARIMA models to the sketch context, we only consider the following two types of ARIMA models (the names are based on the number of differences):

- **ARIMA0**: ARIMA models of order $(p \leq 2, d = 0, q \leq 2)$
- **ARIMA1**: ARIMA models of order $(p \leq 2, d = 1, q \leq 2)$

In ARIMA models, the choice of MA and AR coefficients MA_i ($i = 1, ..., q$) and AR_j ($j = 1, ..., p$) must ensure that the resulting models are *invertible* and *stationary*. As a necessary but insufficient condition, MA_i and AR_j must belong to the range $[-2, 2]$ when $p, q \leq 2$.

3.3 Change detection module

After constructing the forecast error sketch $S_e(t)$, the change detection module chooses an alarm threshold T_A based on the estimated second moment of $S_e(t)$: $T_A \overset{\text{def}}{=} T \cdot [\, \text{ESTIMATEF2}(S_e(t))\,]^{\frac{1}{2}}$, where T is a parameter to be determined by the application.

Now for any key a, the change detection module can reconstruct its forecast error in $S_e(t)$ using $\text{ESTIMATE}(S_e(t), a)$ and raise an alarm whenever the estimated forecast error is above the alarm threshold T_A.

The remaining question is how to obtain the stream of keys for the change detection module. Sketches only support reconstruction of the forecast error associated with a *given* key. It does not contain information about what keys have appeared in the input stream.

There are several possible solutions to this problem. With the brute-force solution, one can record all the keys that appeared in recent intervals (*e.g.*, the same interval t over which $S_e(t)$ is defined) and replay them after $S_e(t)$ has been constructed. This still requires maintaining per-flow information. Its scalability is limited by the maximum number of keys that appear in the window for key collection. We can avoid keeping per-flow state by using a two-pass algorithm—construct $S_e(t)$ in the first pass and detect changes on the second pass. Since the input stream itself will provide the keys, there is no need for keeping per-flow state. This requires access to the same input stream twice and thus useful only in the offline context. A third alternative is to use the keys that appear after $S_e(t)$ has been constructed. This works in both online and offline context. The risk is that we will miss those keys that do not appear again after they experience significant change. This is often acceptable for many applications like DoS attack detection, where the damage can be very limited if a key never appears again. Note that we do not need to do this for every newly arrived item. If we can tolerate the risk of missing some very infrequent keys, we can sample the (future) input streams and only work on a sub-stream of keys. Another possibility is to incorporate combinatorial group testing into sketches [14]. This allows one to directly infer keys from the (modified) sketch data structure without requiring a separate stream of keys. However, this scheme also increases the update and estimation costs and additional research is required to make it more efficient. In this paper, we use the offline two-pass algorithm in all experiments.

3.4 Parameter configuration

Our change detection framework includes sketch-related parameters as well as control parameters for various forecasting models. Below we provide guidelines and heuristics for properly configuring these parameters—an important step for making our framework practical.

3.4.1 Sketch parameters: H and K

There are two sketch-related parameters: the number of hash functions (H), and the size of hash tables (K). Depending on the choice of H and K, *k-ary* sketches can provide probabilistic guarantees on the estimation accuracy of the forecast errors and their total energy (see Appendix A and B for details). We can use such analytical results to determine the choice of H and K that are sufficient to achieve targeted accuracy. As the analytical results apply in a data-independent way, the resulting H and K may be too conservative for the actual dataset. Hence, we use analytical results to derive data-independent choice of H and K and treat them as upper bounds. We then use training data to find the best (data-dependent) H and K values.

3.4.2 Forecasting model parameters

Criteria for good parameters In the context of univariate time series forecasting, a commonly used simple heuristic for configuring model parameters is choosing parameters that minimize the total *residual energy*, *i.e.*, the sum of squares (of forecast errors) over time.

We can extend the above heuristic to the sketch context and look for parameters that minimize the *total energy* in the resulting forecast error sketches over time $\sum_t F_2(S_e(t))$, where $F_2(S_e(t))$ is the second moment for all the forecast errors summarized by sketch $S_e(t)$.

We will not know the *true* $F_2(S_e(t))$ unless we do per-flow analysis for each parameter setting, which can be prohibitive. Instead we use the estimated second moment $F_2^{\text{est}}(S_e(t))$, as long as $F_2^{\text{est}}(S_e(t))$ closely approximates $F_2(S_e(t))$. In other words, we try to find parameters that minimize the estimated total energy of forecast errors $\sum_t F_2^{\text{est}}(S_e(t))$.

Multi-pass grid search For parameters that are continuous, we use a *multi-pass grid search* algorithm to find a good choice. Consider for example the EWMA model. The first pass finds a parameter $\alpha \in \{0.1, 0.2, ..., 1.0\}$ that minimizes the estimated total energy for the forecast errors. Let a_0 be the best α. The second pass equally subdivides range $[a_0-0.1,\ a_0+0.1]$ into $N = 10$ parts and repeats the process. We obtain high precision via multiple passes. For models with integral parameters, such as the moving average model, we can simply vary the parameter to find the best one. Note that grid search is only a heuristic. It does not guarantee that we will find the optimal parameter combination that minimizes the estimated total energy for forecast errors. However, we only need to have good enough parameters such that the resulting model captures the overall time series behavior. We will show later that grid search indeed achieves this.

4. EXPERIMENTAL SETUP

We use large amounts of real Internet traffic data to evaluate and validate our approach. Below we describe our datasets and the experimental parameter settings.

4.1 Dataset description

Input data is chosen to be four hours worth of *netflow* dumps from ten different routers in the backbone of a tier-1 ISP. Nearly 190 million records are processed with the smallest router having 861K records and the busiest one having over 60 million records in a contiguous four hour stretch.

4.2 Experimental parameters

In this section we present the various values of parameters that we used in our experiments and justify their choices. We also present ways in which these values should be tailored in using our approach based on the local data available. Note that some of the parameters would have different values when the sketch technique is used for different applications.

The cost of estimation and updating is dominated by the number of hash tables, so we choose small values for H. Meanwhile, H improves accuracy by making the probability of hitting extreme estimates exponentially small (see Theorem 2, 3, and 5 in Appendix), suggesting again that it is enough to use small H. We vary H to see the impact on the accuracy of estimation with respect to the cost. Our choices of H (1, 5, 9, and 25) are driven by the fact that we can use optimized median networks to find the medians quickly without making any assumptions on the nature of the input [16, 37]. The analytic upper bound needed to provide a specific degree of error threshold by using k-ary sketches is used as the upper reach of K. We can tighten the lower bound of zero by empirically examining values between 0 and the upper bound in log(upper-bound) steps. In our experiments we used an upper bound of 64K and using our data we quickly zoomed in on a lower bound of K = 1024.

Another important parameter is the interval size: a long interval would result in delays since our scheme reports anomalies at the end of each interval and we will miss more events that occur within a single interval. A short interval requires us to update the sketch-based forecasting data structures more frequently. We choose 5 minutes as a reasonable tradeoff between the responsiveness and the computational overhead. Such an interval is used in other SNMP based network anomaly detection systems [3]. We also use 1 minute intervals to examine the impact of shorter intervals.

Each of the six models requires different choices of parameters. For the moving average models (MA and SMA) we pick a single time interval to be the minimum window size and 10 (12) to be the maximum window size for interval size of 5 (1) minutes. The window size yielding the minimum total energy of forecast errors across each of the interval values is selected as the parameter. For the remaining models we apply a 2-pass grid search algorithm to choose different parameters. For the EWMA and NSHW models, during each pass we partition the current ranges into 10 equal intervals. For ARIMA models, however, the number of parameters is much larger and the search space becomes too large if we partition each parameter range into 10 parts. To limit the search space, we partition the current search range into 7 parts instead. During grid search, H is fixed at 1 and K at 8K. As we will see later, with H = 1 and K = 8K, the estimated total energy of forecast errors closely approximates the true energy obtained using per-flow analysis.

5. EXPERIMENT RESULTS

In this section, we present the results of our evaluation of the feasibility of using sketches for change detection. The setup for the various experiments is described in Section 4.2 and we present results in detail for three models (EWMA and ARIMA with d = 0 and 1) and occasional results for NSHW. We should note that in most cases the results from the various models are largely similar and we exclude them in the interest of brevity.

The evaluation is divided into three parts: We first report on the validity of the parameters generated by our grid search. Next, we report on evaluation of sketches at the flow-level—focusing on what sketch reports as (i) the top-N flows with the maximum absolute forecast errors, and (ii) the flows whose absolute forecast error exceeds a threshold, and comparing the sketch report with what per-flow scheme reports. We then report on the implementation complexity and the running time.

5.1 Grid Search

The experiments in this section are concerned with determining appropriate parameter settings for the forecast models, values for H and K, and in evaluating the usefulness of grid search.

- We use the estimated total energy (instead of the *true* total energy) as the metric for selection of the forecast model parameter setting(see Section 3.4.2). For this approach to yield good performance, we need to ensure that the estimated value closely tracks the true value. This is the focus of the first part of the study in this section.
- We also explore the space of (H,K) values and various parameter settings to zoom in on suitable choices of H and K that result in good performance.
- Note that we use grid search to select the parameter setting that results in the minimum total energy (see Section 3.4.2). In this section we evaluate the "goodness" of the parameter selected by grid search, compared to a random selection of parameters.

5.1.1 *Results*

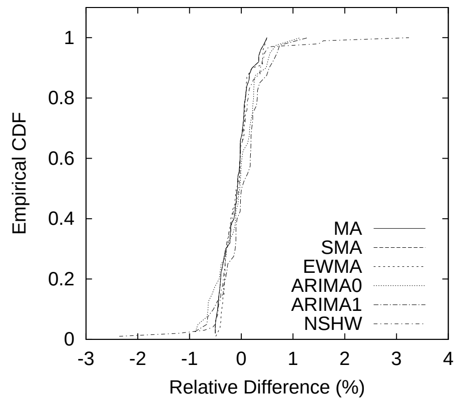
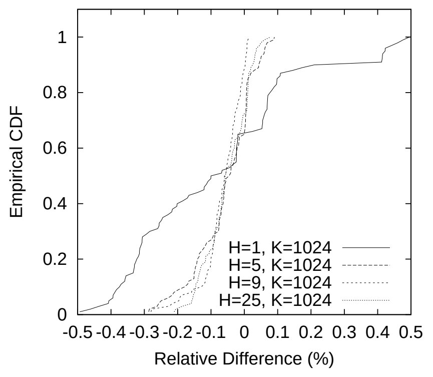
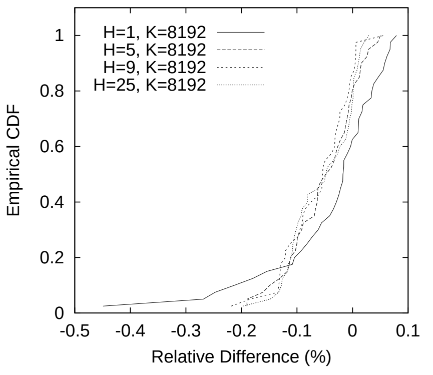


Figure 1: CDF for Relative Difference: all models, interval=300, H=1,K=1024

We first perform a set of experiments (called *random*) over a collection of 10 router files (consisting of over 189 million flow records). For each forecast model, we randomly select a number of points in the model parameter space, and for each chosen point and (H,K) value combination, run both sketch and per-flow based detection on each router trace. The goal here is to examine differences between the different forecast models, and to evaluate parameter value choices for H and K (the hash table and range sizes). The experiment also allows us to explore how sketches and per-flow compare when the forecast parameters are not selected carefully. The comparison metric is the *Relative Difference*, which is defined as: the difference between the total energy (square root of the sum of



(a) Model=EWMA



(b) Model=ARIMA0

Figure 2: Result of varying H in random

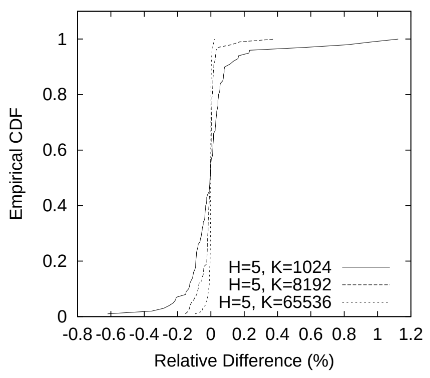
second moments for each time interval) computed from the sketch-based technique and the total energy obtained using per-flow detection, expressed as a percentage of the total energy obtained using per-flow detection. For a particular forecast model and (H,K) combination, for each router file, we obtain multiple Relative Difference values, one for each selected point in the parameter space for that model. In Figures 1-3, each curve corresponds to a particular forecast model and (H,K) combination, and represents the empirical CDF of the Relative Difference values aggregated from across all the routers.

Figure 1 shows that even for small H (1) and K (1024), across all the models, most of the mass is concentrated in the neighborhood of the 0% point on the x-axis, indicating that even for randomly chosen model parameters, the total energy from the sketch-based approach is very close to that for per-flow. Only for the NSHW model a small percentage of points have sketch values that differ by more than 1.5% from the corresponding per-flow values. The worst case difference is 3.5%.

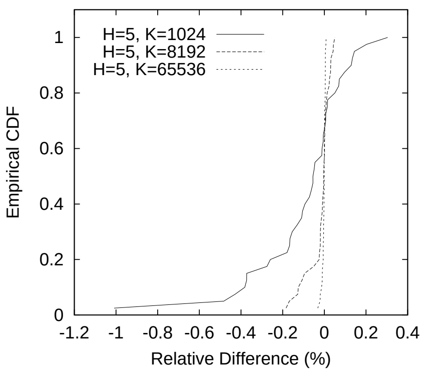
Next, we examine the impact of varying the H parameter. Figure 2 shows, for the EWMA and ARIMA0 models, that there is no need to increase H beyond 5 to achieve low relative difference.

The last set of results for the random parameter technique is shown in Figure 3, and demonstrates that once K = 8192 (8K) the relative difference becomes insignificant, obviating the need to increase K further.

The grid search technique for identifying parameters uses six models for both 60s and 300s intervals, a representative sample of router files (one large, one medium, and one small sized file), and



(a) Model=EWMA



(b) Model=ARIMA0

Figure 3: Result of varying K in random

(H=1, K=8192) combination. For each (model,router,H,K) combination, grid search outputs the parameter value(s) for that model that minimize the total energy in the resulting forecast errors. Using this parameter setting output by grid search, we run per-flow analysis and obtain the corresponding total energy. The per-flow estimate is then compared against the per-flow estimates of the random parameters generated in the previous technique, for the same router file and model. The goal of this experiment is twofold: first, we ensure that grid search results are *never* worse than any of the per-flow values of the random parameters. Second, we show that grid search results can be significantly better than the results in the random case. Our experimental results (no graphs shown) show that in all cases (all models, three router files, both intervals) grid search is never worse than the random parameters. Secondly, in at least 20% of the cases the results with the random parameters are at least twice (and in many cases much more) as bad as the errors in the grid search case. This justifies the use of grid search to generate the parameters for the remainder of the experiments.

5.2 Accuracy

After validating the set of parameters from the grid search scheme, we need to demonstrate that the sketch scheme's results are accurate as compared to per-flow estimates. We estimate accuracy in our experiments via two techniques: (i) Top-N, and (ii) Thresholding.

The values of H and K are key to the accuracy of our forecasts as well as efficient operation. Care must be taken to choose the proper range of values on a per-application basis. Our experimental results based on large and diverse data sets show that the values we have chosen (H = 1..25), (K = 1K .. 64K) are indeed the right ones for the change detection class of applications.

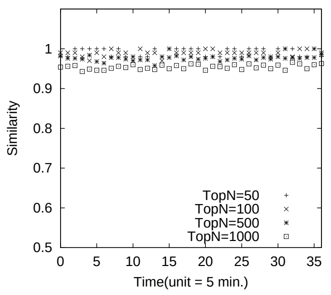
5.2.1 *Top-N sketch vs. per-flow evaluation*

Here we are interested in exploring, for a given N, how many of the top-N flows (ranked in order of decreasing magnitude of forecasting errors) detected by the per-flow scheme are also detected as top-ranking by our sketch-based scheme.

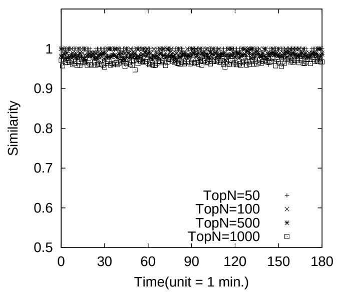
We choose three values of H (5, 9, 25) and K (8K, 32K, 64K), two values of intervals (60s and 300s), and selected three router data files representing high volume (over 60 Million), medium (12.7 Million), and low (5.3 Million) records, to carry out sketch accuracy evaluation across all models. For the model parameters, we use the parameter values selected by the grid search process. For each time interval, we generate the top-N flows with the maximum absolute forecast errors (recall that a higher absolute forecast error indicates that a flow's volume has a higher deviation from that predicted by the underlying model) for both sketches and per-flow techniques. For four values of N (50, 100, 500, 1000), we see how many of the top-N flows are in common between the two resulting sets and compute a similarity metric N_{AB}/N, where N_{AB} is the number of common elements in the two sets.

While some of the top-N ranked elements from the per-flow technique may not belong to exactly the top-N elements output by the sketch technique, the hope is that these elements will still be high in the sketch-based ranking. Thus, it is possible to increase the accuracy by comparing the top-N per-flow list with additional elements in the sketch-based ranked list. To evaluate this possibility, a second set of comparisons involves comparing the top-N per-flow results against the top-X*N (X = 1, 1.2, 1.5, 2) results from the sketch-based approach.

5.2.1.1 *Results.*



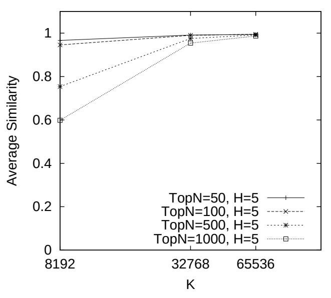
(a) Interval=300



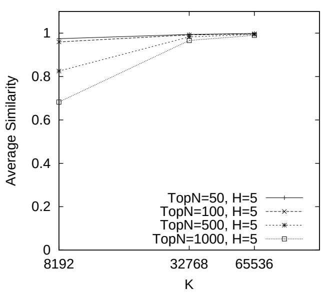
(b) Interval=60

Figure 4: Overall similarity of sketch and per-flow for large router file, H=5, K=32K

Results in this section show how well sketches perform when compared with per-flow by comparing their top-N (N=50, 100, 500, 1000) flows. The metric is essentially a similarity one: the number of common elements in the two sets normalized by N. We start by showing how this metric is remarkably consistent across the time intervals, for moderate H and K. We set aside the first hour of the four hour data sets for model warmup purposes leaving 180 and 37 intervals respectively in the 60s and 300s time interval cases. Figures 4 (a) and (b) show that even for large N (1000), the similarity is around 0.95 for both the 60s and 300s intervals, for H=5 and K=32K. In the rest of the figures in this section, we show the mean similarity value across the 180 and 37 intervals.

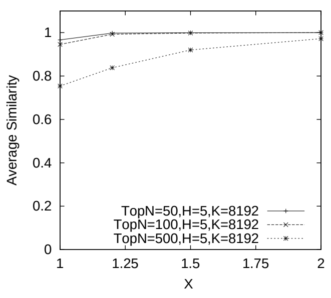


(a) Interval=300

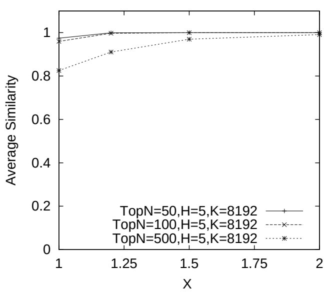


(b) Interval=60

Figure 5: Similarity using EWMA model for large router file (topN sketch vs. topN per-flow)

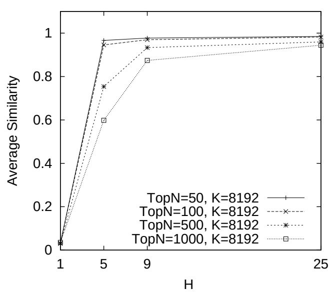


(a) Interval=300

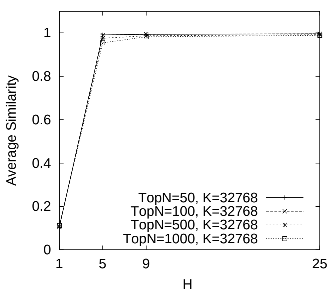


(b) Interval=60

Figure 6: topN vs. top-X*N for EWMA model for large router file

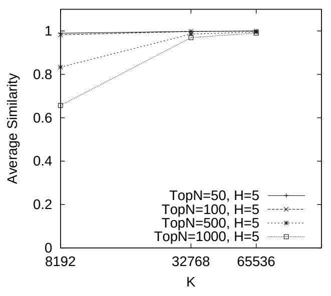


(a) Interval=300

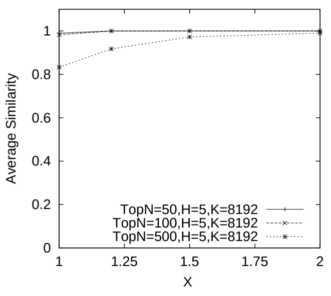


(b) Interval=60

Figure 7: Effect of varying H and K for EWMA model for large router file (topN sketch vs. topN per-flow)

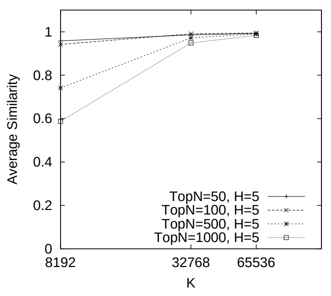


(a) Interval=300

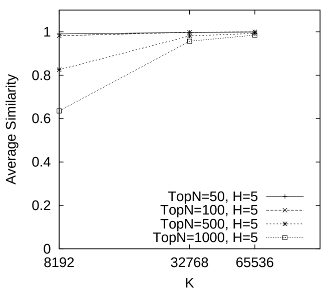


(b) Interval=60

Figure 8: Similarity metric for EWMA model for medium sized router file.(a) top-N vs. topN (b)top-N vs. top-X*N



(a) Router=large



(b) Router=Medium

Figure 9: Similarity metric for model ARIMA 0 for large and medium sized router files (top-N vs. topN)

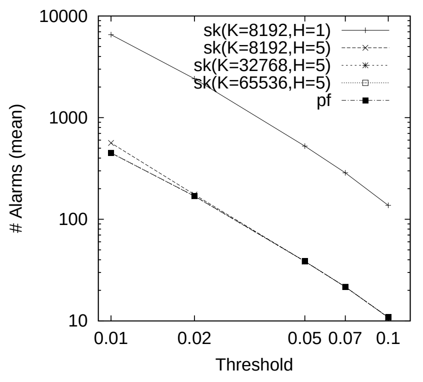
Figures 5 (a) and (b) use the EWMA model to show average similarity (across the time intervals), where H is fixed at 5 and K varies between 8K and 64K, for both 300s and 60s time intervals. As can be seen, for K=32K, the similarity is over 0.95 even for large N. For a smaller N (say 50 or 100), the overlap is nearly 100%. Larger values of K are of limited additional benefit. We note that similarity improves (for large N) with the smaller interval size of 60. This increased accuracy can be attributed to the fact that for a smaller interval, there are potentially fewer flows that have to be summarized in each interval.

We next explore the potential of improving the accuracy by performing the top-N vs. top-X*N (X = 1, 1.2, 1.5, 2) comparison. As can be seen in Figures 6 (a) and (b), the similarity increases for K=8K, even for large N. With X=1.5, the similarity has risen significantly even for large N. For the settings we examined, we get very high accuracy with $X \leq 1.5$, and higher values of X result in marginal additional accuracy gains. This is desirable, because a larger X while increasing accuracy, also results in more false positives.

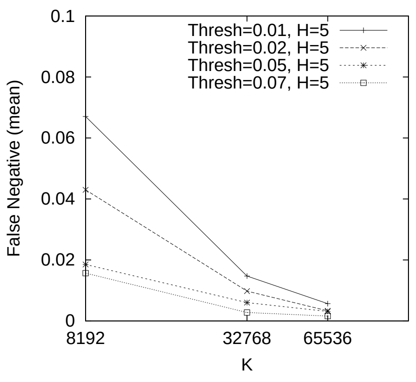
We next consider the effect of varying H on the accuracy. Figure 7 (a) shows that with a small K=8K, H needs to be at least 9 to get high similarity values, especially for large N. A large H is undesirable as an increase in H directly corresponds to increased computation overhead (the number of update operations per key is proportional to the value of H) and memory (for sketches) overhead. But, as Figure 7 (b) shows, even for very large N, increasing K to 32K instantly increases similarity to nearly 1, for a small H=5. A larger K (for sketches) implies a large space overhead. This suggests a space-computation overhead tradeoff. In many applications where the computation overhead is more critical, with K = 32K or more, we can get good accuracy results with small H.

We next show the results for a different router file (all files have similar output). Figure 8 (a) and (b) show the similarity metric for EWMA model for the medium sized router file.

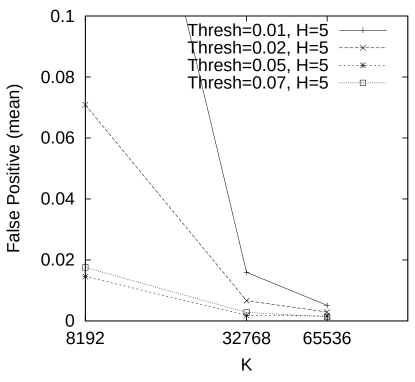
Likewise, we show the effect of another model (all models had similar results)—this time we use ARIMA0, *i.e.*, ARIMA with $d = 0$. Figure 9 (a) and (b) show similarity for large and medium sized router files respectively for an interval of 300s.



(a) Interval=60s, Use of thresholds

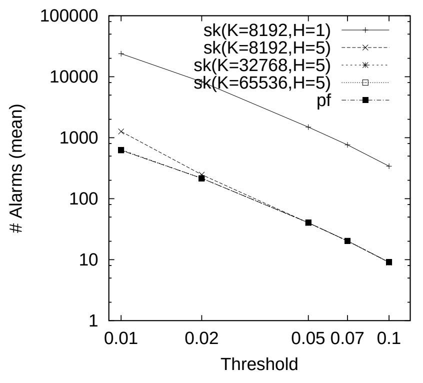


(b) Interval=60s, False negatives

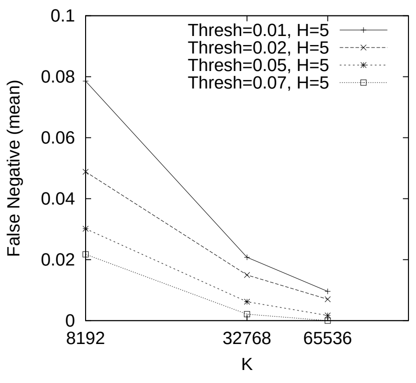


(c) Interval=60s, False positives

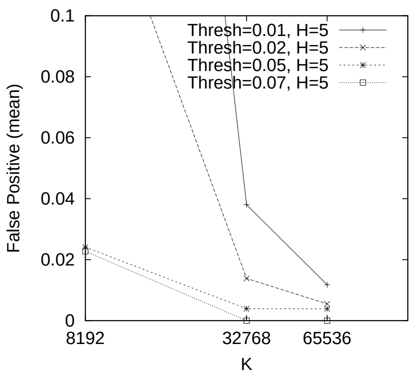
Figure 10: Thresholding: large router, 60s interval, non-seasonal Holt-Winters model



(a) Interval=300s, Use of thresholds



(b) Interval=300s, False negatives



(c) Interval=300s, False positives

Figure 11: Thresholding: large router, 300s interval,non-seasonal Holt-Winters model

5.2.2 Topthresh sketch vs. per-flow evaluation

Instead of comparing just the top-N values, as in the previous accuracy tests, we now limit the flows to ones whose absolute forecast error is greater than or equal to a certain fraction of the L2 norm (square root of the sum of squares of the forecast errors of all flows in a time interval). We vary this threshold level across 0.01, 0.02, 0.05, 0.07, and 0.1. We show results for each of the two time intervals (60s, 300s) for three models (EWMA, NSHW, and ARIMA with $d = 0$). For each of sketch and per-flow based change detection, we rank the flows in decreasing order of absolute value of forecast error.

The metrics of interest here are the false negative ratio, false positive ratio, and the number of alarms. For a given value of threshold τ, let $N_{pf}(\tau)$ and $N_{sk}(\tau)$ refer to the number of flows that meet the threshold in per-flow and sketch based detection, respectively. The number of alarms for per-flow and sketches are then $N_{pf}(\tau)$ and $N_{sk}(\tau)$ respectively. Let $N_{AB}(\tau)$ be the count of flows that are common to both the sketch and per-flow lists. The false negative ratio is computed as $\frac{N_{pf}(\tau)-N_{AB}(\tau)}{N_{pf}(\tau)}$. The false positive ratio is: $\frac{N_{sk}(\tau)-N_{AB}(\tau)}{N_{sk}(\tau)}$. At this point, for each metric we have a time series, with one value per time interval. In our study below, we consider the mean value over the entire time series.

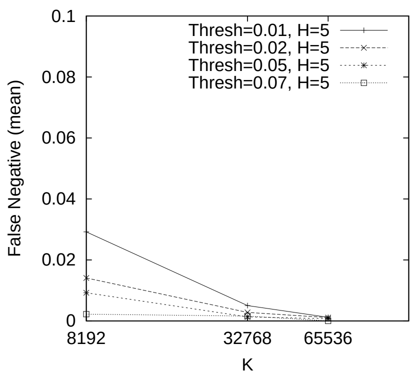
5.2.2.1 Results.

In this part of the results, we demonstrate the similarity of sketch and per-flow results when flows are selected by thresholding. The overall summary here is that with K set to be at least 32K, we can provide excellent guarantees for low false negatives and false positives. We show two sets of figures.

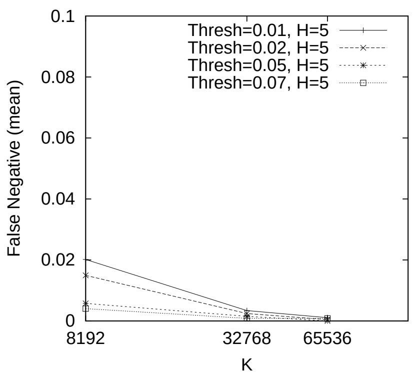
The first set is for the large sized router data file and uses the non-seasonal Holt-Winters model for the 60s (Figure 10) and 300s (Figure 11) time intervals. Figure 10(a) shows that for a very low value of H (=1), the number of alarms are very high. Simply increasing H to 5 suffices to dramatically reduce the number of alarms. The figure also demonstrates the significant reduction in number of alarms that can be realized by increasing the threshold value. Finally, it shows that there is virtually no difference between the per-flow results and the sketch results when $H \geq 5$ and $K \geq 8K$.

Figure 10(b) shows that for K=32K and beyond, the false negative ratio drops rapidly to be less than 2% even for very low threshold values and is below 1% for threshold of 0.05. The false positive ratio, as Figure 10(c) shows, for K=32K and even a low threshold of 0.02, is quite low (below 1%). The overall results are similar for the 300s interval.

The second set of figures uses the medium sized router data file, for a single interval size (300s) and varies across four mod-

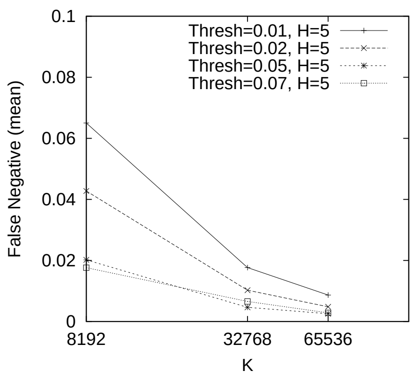


(a) Model=EWMA

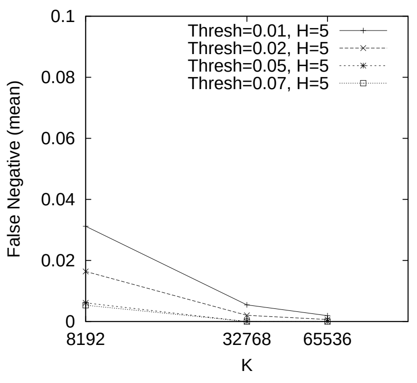


(b) Model=NSHW

Figure 12: Thresholding false negatives: medium router, 300s interval, EWMA, NSHW models



(a) Model=ARIMA0



(b) Model=ARIMA1

Figure 13: Thresholding false negatives: medium router, 300s interval, ARIMA models

els: EWMA, non-seasonal Holt-Winters model, ARIMA with d = 0 and 1. We show only the false negative and false positive ratios.

Figure 12 (a) shows the false negative ratio for the EWMA model to be well below 1% for thresholds larger than 0.01. Likewise, Figure 12 (b) shows the false negative ratio for the non-seasonal Holt-Winters model to be slightly better than the EWMA model.

Figure 13 (a) and (b) show for the two different ARIMA models (d = 0 and 1, respectively), that false negatives are low but differ a bit as compared to EWMA and NSHW models for a low threshold of 0.01.

Figure 14 (a) and (b) show the false positive ratios for the EWMA and NSHW models respectively, to be well below 1% for thresholds larger than 0.01 for K=32K or higher. Likewise, Figure 15 (a) and (b) show low false positive ratios for both ARIMA models.

5.3 Implementation complexity and performance

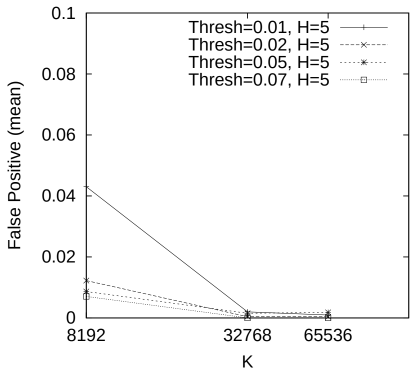
There are three key components in our sketch-based change detection implementation: 4-universal hash functions, sketches, and forecasting. The implementation of 4-universal hash functions is about 200 lines, sketches around 250 lines, while forecasting code varies with the forecasting models and all of the models together are less than 800 lines (all in C).

Hash computation and sketch UPDATE need to be done on every item in the input stream. Sketch ESTIMATE, by default, also needs to be done on a per-item basis. However, if we are willing to miss some keys that appear too infrequently (which arguably can only cause limited damage) we can sample the stream of incoming keys and only do ESTIMATE on the substream. Operations like ESTIMATEF2 only need to be done infrequently—once every interval—and their amortized costs are insignificant.

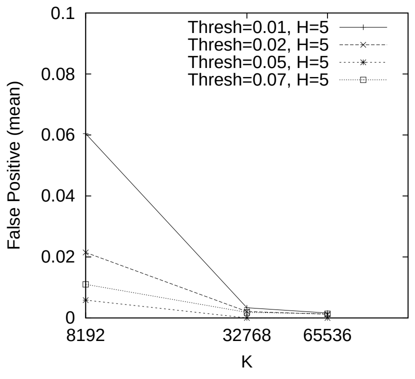
operations	running time (sec)	
	computer A	computer B
compute 8 16-bit hash values	0.34	0.89
UPDATE ($H = 5$, $K = 2^{16}$)	0.81	0.45
ESTIMATE ($H = 5$, $K = 2^{16}$)	2.69	1.46

Table 1: Running time for performing 10 million hash computations and sketch operations on computer A (400 MHz SGI R12k processor running IRIX64 6.5) and B (900 MHz Ultrasparc-III processor running Solaris 5.8).

Table 1 summarizes the running time for performing 10 million hash computations, sketch UPDATE, and sketch ESTIMATE operations. Each hash computation produces 8 independent 16-bit hash values and therefore suffices for *k-ary* sketches with $H \leq 8$ and $K \leq 2^{16}$. Both UPDATE and ESTIMATE assume the hash values

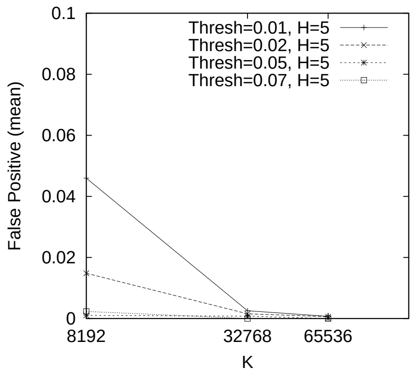


(a) Model=EWMA

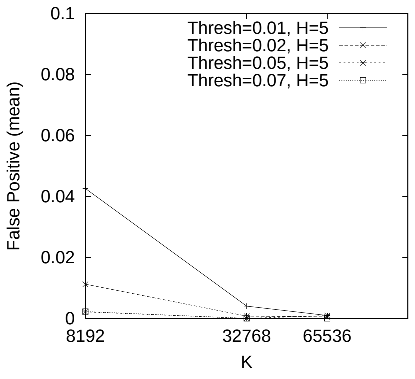


(b) Model=NSHW

Figure 14: False positives: medium router, 300s interval, EWMA, NSHW models



(a) Model=ARIMA0



(b) Model=ARIMA1

Figure 15: False positives: medium router, 300s interval, ARIMA models

have already been computed (which needs to done only once per item). The sketch parameters we use are $H = 5$ and $K = 2^{16}$. As we can see, the overhead of these operations are not very high. We note that the current implementation has not been fully optimized allowing room for further speedup.

6. ONGOING WORK

Our preliminary exploration indicates that our sketch-based change detection method is highly accurate when compared with per-flow time series analysis. It offers promise to be a building block for network anomaly detection and traffic measurement. We outline some avenues that we are exploring

- *Online change detection*: We have currently evaluated our methods in an offline setting. These evaluations suggest that the technique may be capable of near real-time change detection. One change required is modifying our technique to obtain the forecast model parameters online. One possible way is periodically recomputing the forecast model parameters using history data to keep up with changes in overall traffic behavior.

- *Avoiding boundary effects due to fixed interval sizes*. Possible solutions include (i) simultaneously run multiple models using different interval sizes, and different starting points, (ii) randomize the interval size (*e.g.*, using exponentially distributed interval size) and detect changes of total values normalized by interval size. The linearity of sketches makes this possible.

- *Reducing false positives*: We have focused on accurate detection of significant deviation from normal behavior. However, some anomalies are benign. The problem of reducing false alarms is a major challenge for all change-detection based network anomaly detection systems. Our change detection framework has tunable parameters which can be adjusted to limit the false positives. For instance, the technique can be asked to only report the top N major changes, or the changes that are above a threshold. The particular application needs will guide the actual setting of these tunable parameters.

- *Combining with sampling*: Given the massive volumes of data generated in large networks, sampling is increasingly being used in ISP network measurement infrastructures for reducing the volume of data that has to be collected. Our current approach combines time series analysis with sketches for scalable change detection in massive data sets. We plan to explore combining sampling techniques with our approach for increased scalability.

- *Better guidelines for choosing parameters*: Given the wide range of parameters we have, it would be useful to have reasonable guidance for selecting proper and justifiable values for them. The full factorial method [5] in the statistical experimental design domain can help in narrowing the number of levels (or "versions") for the various variables. We are exploring such techniques to see which parameters are independent of each other and move towards identifying reasonable values overall based on the similarity. For example, H has overall impact independent of other parameters. The tedium related to having multiple runs can also be reduced for example by using Yates algorithm [5].

7. SUMMARY

In this paper, we presented a *sketch-based change detection* technique. Our work is motivated by anomaly detection and other applications that can benefit from having a quick and efficient change detection mechanism. The scheme is capable of detecting significant changes in massive data streams with a large number of network time series. As part of the technique, we designed a variant of the sketch data structure, called *k-ary sketch*, which uses a constant, small amount of memory, and has constant per-record update and reconstruction cost. We implemented a variety of time series forecast models (ARIMA, Holt-Winters, etc.) on top of such summaries and detect significant changes by looking for flows with large forecast errors. We also presented heuristics for automatically configuring the forecast model parameters.

We demonstrate using a large amount of real Internet traffic data that our sketch-based change detection method is highly accurate when compared with per-flow analysis, and can be implemented at low computation and memory costs. Our preliminary results are promising and point to the potential of using our technique as a building block for network anomaly detection and traffic measurement in large networks.

8. REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[2] H. Arsham. Time series analysis and forecasting techniques. http://obelia.jde.aca.mmu.ac.uk/resdesgn/arsham/opre330Forecast.htm.

[3] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

[4] P. Barford and D. Plonka. Characteristics of network traffic flow anomalies. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, San Francisco, CA, November 2001.

[5] G. E. P. Box, W. G. Hunter, and J. S. Hunter. *Statistics for Experimenters*. John Wiley, 1978.

[6] G. E. P. Box and G. M. Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, 1976.

[7] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis, Forecasting and Control*. Prentice-Hall, Englewood Cliffs, 1994.

[8] P. Brockwell and R. Davis. *Introduction to Time Series and Forecasting*. Springer, 1996.

[9] J. Brutlag. Aberrant behavior detection in time series for network monitoring. In *Proc. USENIX LISA XIV*, New Orleans, LA, December 2000. http://www.usenix.org/events/lisa2000/full_papers/brutlag/brutlag_html/index.html.

[10] J. Carter and M. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

[11] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proc. of ICALP 2002*, pages 693–703, 2002. http://www.cs.princeton.edu/˜moses/papers/frequent.ps.

[12] C. Chen and L.-M. Liu. Forecasting time series with outliers. *Journal of Forecasting*, 12:13–35, 1993.

[13] C. Chen and L.-M. Liu. Joint estimation of model parameters and outlier effects in time series. *Journal of the American Statistical Association*, 88:284–297, 1993.

[14] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. In *Proc. ACM PODC '2003*, July 2003.

[15] M. Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. Technical Report 2001-21, DIMACS Technical Report, November 2001.

[16] N. Devillard. Fast median search: an ansi c implementation, July 1998. http://ndevilla.free.fr/median/median.pdf.

[17] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proc. ACM SIGCOMM '2002*, Pittsburgh, PA, August 2002.

[18] F. Feather, D. Siewiorek, and R. Maxion. Fault detection in an ethernet network using anomaly signature matching. In *Proc. ACM SIGCOMM '93*, 1993.

[19] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. ACM SIGCOMM '00*, August 2000.

[20] K. Fox, R. Henning, J. Reed, and R. Simonian. A neural network approach towards intrusion detection. Technical report, Technical Report, Harris Corporation, July 1990.

[21] A. C. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. J. Strauss. Quicksand: Quick summary and analysis of network data. Technical Report 2001-43, DIMACS Technical Report, November 2001.

[22] C. Hood and C. Ji. Proactive network fault detection. In *Proc. IEEE INFOCOM '97*, Kobe, Japan, April 1997.

[23] K. J. Houle, G. M. Weaver, N. Long, and R. Thomas. Trends in Denial of Service Attack Technology. http://www.cert.org/archive/pdf/DoS_trends.pdf.

[24] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proc. of the 41st Symposium on Foundations of Computer Science*, 2000.

[25] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In *Proceedings of the World Wide Web Conference*, Honolulu, Hawaii, May 2002. http://www.research.att.com/˜bala/papers/www02-fc.html.

[26] I. Katzela and M. Schwartz. Schemes for fault identification in communication networks. *IEEE/ACM Transactions on Networking*, 3(6):753–764, December 1995.

[27] M. J. Lebo and W. H. Moore. Foreign policy behavior and fractional integration. *Journal of Conflict Resolution*, 1(47):13–32, February 2003. http://garnet.acns.fsu.edu/˜whmoore/research/Lebo&Moore2003.pdf.

[28] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. The Spread of the Sapphire/Slammer Worm. Technical report, Technical Report, February 2003. http://www.cs.berkeley.edu/~nweaver/sapphire/.

[29] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. In *Proc. of the USENIX Security Symposium*, Washington D.C., August 2001. http://www.cs.ucsd.edu/~savage/papers/UsenixSec01.pdf.

[30] S. Muthukrishnan. Data streams: Algorithms and applications, 2003. Manuscript based on invited talk from *14th SODA*. Available from http://www.cs.rutgers.edu/~muthu/stream-1-1.ps.

[31] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23–24):2435–2463, December 1999. ftp://ftp.ee.lbl.gov/papers/bro-CN99.ps.gz.

[32] M. Roesch. Snort – Lightweight Intrusion Detection for Networks. In *Proc. USENIX Lisa '99*, Seattle, WA, November 1999.

[33] M. Thorup and Y. Zhang. Tabulation based 4-universal hashing with applications to second moment estimation, 2003. Under submission. Available from http://www.research.att.com/~yzhang/papers/hash-tm03.ps.

[34] J. Toelle and O. Niggemann. Supporting intrusion detection by graph clustering and graph drawing. In *Proc. RAID '2000*, Toulouse, France, October 2000.

[35] R. S. Tsay. Time series model specification in the presence of outliers. *Journal of the American Statistical Association*, 81:132–141, 1986.

[36] R. S. Tsay. Outliers, level shifts, and variance changes in time series. *Journal of Forecasting*, 7:1–20, 1988.

[37] T.S.Huang, G. J. Yang, and G. Y. Tang. A fast two-dimensional median filtering algorithm. *IEEE transactions on acoustics, speech and signal processing*, 27(1), February 1979.

[38] A. Ward, P. Glynn, and K. Richardson. Internet service performance failure detection. *Performance Evaluation Review*, August 1998.

[39] M. Wegman and J. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22:265–279, 1981.

[40] N. Ye. A markov chain model of temporal behavior for anomaly detection. In *Workshop on Information Assurance and Security*, West Point, NY, June 2000.

APPENDIX

Notation For any $a, b \in [u]$, let $a \sim b$ denote $h(a) = h(b)$, $a \not\sim b$ denote $h(a) \neq h(b)$.

A. ANALYSIS FOR V_A ESTIMATION

Accuracy of $v_a^{h_i}$ The following theorem states that each $v_a^{h_i}$ ($i \in [H]$) is an unbiased estimator of v_a with variance inversely proportional to $(K-1)$.

THEOREM 1.

$$\mathrm{E}\left[v_a^{h_i}\right] = v_a \tag{1}$$

$$\mathrm{Var}\left[v_a^{h_i}\right] \leq \frac{F_2}{K-1} \tag{2}$$

PROOF. For any $h \in \{h_0, ..., h_{H-1}\}$, we have

$$\begin{aligned} v_a^h &= \frac{\sum_{b\sim a} v_b - (1/K)\cdot \sum_b v_b}{1-1/K} \\ &= \sum_{b\sim a} v_b - \frac{1}{K-1}\sum_{b\not\sim a} v_b \\ &= v_a + \sum_{b\sim a \wedge b\neq a} v_b - \frac{1}{K-1}\sum_{b\not\sim a} v_b \end{aligned} \tag{3}$$

Define

$$X_{a,b} = \begin{cases} 1 & \text{if } b \sim a \\ -\frac{1}{K-1} & \text{otherwise} \end{cases}$$

(3) becomes

$$v_a^h = v_a + \sum_{b\neq a} v_b\, X_{a,b} \tag{4}$$

Since h is 4-universal, for any distinct $a, b \in [u]$, we have

$$\mathrm{E}\left[X_{a,b}\right] = 0 \tag{5}$$

$$\mathrm{E}\left[X_{a,b}^2\right] = \frac{1}{K-1} \tag{6}$$

In addition, for any distinct $a, b, c \in [u]$, we have

$$\mathrm{E}\left[X_{a,b}\, X_{a,c}\right] = 0 \tag{7}$$

Now we are ready to prove the theorem.

$$\mathrm{E}\left[v_a^h\right] =_{(4)} v_a + \sum_{b\neq a} v_b \mathrm{E}\left[X_{a,b}\right] =_{(5)} v_a$$

$$\begin{aligned} \mathrm{Var}\left[v_a^h\right] &= \mathrm{E}\left[\left(v_a^h - \mathrm{E}\left[v_a^h\right]\right)^2\right] \\ &=_{(4)\ (1)} \mathrm{E}\left[\left(\sum_{b\neq a} v_b\, X_{a,b}\right)^2\right] \\ &= \sum_{b\neq a} v_b^2 \mathrm{E}\left[X_{a,b}^2\right] + \sum_{a,b,c \text{ distinct}} v_b v_c\, \mathrm{E}\left[X_{a,b} X_{a,c}\right] \\ &=_{(6)\ (7)} \frac{1}{K-1}\sum_{b\neq a} v_b^2 \leq \frac{F_2}{K-1} \end{aligned}$$

■

Accuracy of v_a^{est} v_a^{est} further improves accuracy by avoiding the extreme estimates. Theorem 2 and 3 summarize the accuracy guarantee of v_a^{est}.

THEOREM 2. *For any $a \in [u]$, $T \in (0,1)$, and $\alpha \in [1, \infty)$, if $|v_a| \geq \alpha\, T\sqrt{F_2}$, then*

$$\Pr\left\{|v_a^{\text{est}}| \leq T\sqrt{F_2}\right\} \leq \left[\frac{4}{(K-1)(\alpha-1)^2\, T^2}\right]^{H/2} \tag{8}$$

PROOF. For any $h \in \{h_0, ..., h_{H-1}\}$, by the Chebyshev inequality, we have

$$\begin{aligned}
&\Pr\left\{|v_a^h| \le T\sqrt{F_2}\right\} \le \Pr\left\{|v_a^h - v_a| \ge |v_a| - T\sqrt{F_2}\right\} \\
\le\ & \Pr\left\{|v_a^h - v_a| \ge (\alpha - 1)\, T\sqrt{F_2}\right\} \\
=\ & \Pr\left\{|v_a^h - \mathrm{E}\left[v_a^h\right]| \ge (\alpha - 1)\, T\sqrt{F_2}\right\} \\
\le\ & \frac{\mathrm{Var}\left[v_a^h\right]}{\left[(\alpha-1)\, T\sqrt{F_2}\right]^2} \qquad \text{(Chebyshev Inequality)} \\
\le\ & \frac{F_2/(K-1)}{\left[(\alpha-1)\, T\right]^2\, F_2} = \frac{1}{(K-1)(\alpha-1)^2\, T^2}
\end{aligned}$$

Since v_a^{est} is obtained by taking the median of H copies of v_a^h, by the Chernoff inequality, we immediately have (8). Both Chebyshev and Chernoff inequalities can be found in [1]. ■

THEOREM 3. *For any* $a \in [u]$, $T \in (0,1)$, *and* $\beta \in [0,1]$, *if* $|v_a| \le \beta\, T\sqrt{F_2}$, *then*

$$\Pr\left\{|v_a^{\mathrm{est}}| \ge T\sqrt{F_2}\right\} \le \left[\frac{4}{(K-1)(1-\beta)^2\, T^2}\right]^{H/2} \tag{9}$$

PROOF. The proof is almost identical and is omitted here in the interest of brevity. ■

As an example, let $K = 2^{16}$, $\alpha = 2$, $\beta = 0.5$, $T = 1/32$, and $H = 20$. If we raise an alarm whenever $v_a^{\mathrm{est}} \ge \sqrt{F_2}/32$, then according to Theorem 2, the probability that we will miss a $v_a > \sqrt{F_2}/16$ is less than 9.0×10^{-13}; according to Theorem 3, the probability that we will falsely raise an alarm for a $v_a < \sqrt{F_2}/64$ is less than 9.5×10^{-7}.

B. ANALYSIS FOR F_2 ESTIMATION

Accuracy of $F_2^{h_i}$ The following theorem is proved in [33], which shows that each $F_2^{h_i}$ forms an unbiased estimator of F_2 with variance inversely proportional to $(K-1)$. [33] also shows that in order to achieve the same variance using count sketch, one has to either live with lower speed or double the memory requirement.

THEOREM 4.

$$\mathrm{E}\left[F_2^{h_i}\right] = F_2 \tag{10}$$

$$\mathrm{Var}\left[F_2^{h_i}\right] \le \frac{2}{K-1} F_2^2 \tag{11}$$

Accuracy of F_2^{est} F_2^{est} further improves accuracy by avoiding the extreme estimates. Theorem 5 provides the accuracy guarantee of F_2^{est}.

THEOREM 5. *For any* $\lambda > 0$, *we have*

$$\Pr\left\{|F_2^{\mathrm{est}} - F_2| > \lambda F_2\right\} \le \left[\frac{8}{(K-1)\lambda^2}\right]^{H/2} \tag{12}$$

PROOF. By Theorem 4 and the Chebyshev inequality,

$$\begin{aligned}
&\Pr\left\{|F_2^{h_i} - F_2| > \lambda F_2\right\} \le \frac{\mathrm{Var}\left[F_2^{h_i}\right]}{(\lambda F_2)^2} \\
\le\ & \frac{2\, F_2^2/(K-1)}{(\lambda\, F_2)^2} = \frac{2}{(K-1)\lambda^2}
\end{aligned}$$

Since F_2^{est} is the median of H copies of $F_2^{h_i}$, by the Chernoff inequality, we immediately obtain (12). ■

As an example, let $K = 2^{16}$, $\lambda = 0.05$, and $H = 20$, Theorem 5 states that the probability that our estimate F_2^{est} is 5% off its real value F_2 is below 7.7×10^{-14}.