STREAMING TECHNIQUES FOR STATISTICAL MODELING

BY YIHUA WU

A dissertation submitted to the
Graduate School—New Brunswick
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
Graduate Program in Computer Science

Written under the direction of
Prof. S. Muthukrishnan
and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October, 2007

ABSTRACT OF THE DISSERTATION

Streaming Techniques for Statistical Modeling

by Yihua Wu

Dissertation Director: Prof. S. Muthukrishnan

Streaming is an important paradigm for handling high-speed data sets that are too large to fit in main memory. Prior work in data streams has shown how to estimate simple statistical parameters, such as histograms, heavy hitters, frequent moments, etc., on data streams. This dissertation focuses on a number of more sophisticated statistical analyses that are performed in near real-time, using limited resources.

First, we present how to model stream data parametrically; in particular, we fit hierarchical (binomial multifractal) and non-hierarchical (Pareto) power-law models on a data stream. It yields algorithms that are fast, space-efficient, and provide accuracy guarantees. We also design fast methods to perform online model validation at streaming speeds.

The second contribution of this dissertation addresses the problem of modeling an individual's behaviors via "signature" for nodes in communication graphs. We develop a formal framework for the usage of signatures on communication graphs and identify fundamental properties that are natural to signature schemes. We justify these properties by showing how they impact a set of applications. We

then explore several signature schemes in our framework and evaluate them on real data in terms of these properties. This provides insights into suitable signature schemes for desired applications.

Finally, the dissertation studies the detection of changes in models on data with unknown distributions. We adapt the sound statistical method of sequential probability ratio test to the online streaming case, without independence assumption. The resulting algorithm works seamlessly without window limitations inherent in prior work, and is highly effective at detecting changes quickly. Furthermore, we formulate and extend our streaming solution to the local change detection problem that has not been addressed earlier.

As concrete applications of our techniques, we complement our analytic and algorithmic results with experiments on network traffic data to demonstrate the practicality of our methods at line speeds, and the potential power of streaming techniques for statistical modeling in data mining.

Acknowledgements

I would like to thank my advisor, Prof. S. Muthukrishnan, for his guidance throughout my Ph.D. studies. Meeting with Muthu is alike to drink from a fire hose, where it is so common for me to spend a week or longer unraveling the new ideas (or suggestions) and key words from Muthu during a single one-hour meeting. Muthu's greatest gift as an advisor is to encourage me and to instill in me the confidence in doing fun research on my own, without allowing me to settle for doing less than the best I can. An excellent speaker himself, I learned from Muthu how to give good talks, which I found extremely useful. On the other hand, he is so effective in overcoming my anxiety, as a friend.

I am fortunate to have industrial collaborations with researchers from several labs. These experiences are eye-openers to me, offering me a unique opportunity to see how people outside universities do "real" work and do research. I am grateful to Muthu for such hookups. I thank Flip Korn for being a thoughtful mentor at AT&T Research. He helped me in every aspect. I benefited greatly from many detailed discussions with him in many ways: in person, via emails and on phone. His ideas and advice helped me find solutions to research problems as well as the path to become a researcher. I am also thankful to Eric van den Berg for his helpful advice on Chapter 5, to Graham Cormode for his consistent interests in and encouragement to my work since my first research project. I am very happy to have collaboration with Graham on Chapter 4.

I want to thank the remaining members of my thesis committee for their time and interest in this dissertation. They are Prof. David Madigan, Prof. Richard Martin, and Dr. Divesh Srivastava.

I spent one summer at AT&T Research, as an intern. I enjoyed all my experiences and I thank researchers there for spending time in talking about my work: Tamraparni Dasu, Marios Hadjieleftheriou, Theodore Johnson, Yehuda Koren, Shubho Sen, Oliver Spatscheck, Divesh Srivastava, Mikkel Thorup, Simon Urbanek, Suresh Venkatasubramanian, Chris Volinsky, and the fellow interns: Emiran Curtmola, Bing Tian Dai, Irina Rozenbaum, Vladislav Shkapenyuk, Hanghang Tong, Ranga Vasudevan, Ying Zhang. I am also thankful to my friends and fellow students in the department, who have made my time at Rutgers colorful.

Finally, I must thank my family members. Their love helped me through my darker moods while in graduate school. Without their supports, none of this would be possible.

Dedication

To my parents: Longcheng Wu and Sufang Xia. Without their selfless love, I could not achieve anything.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

Database research has sought to scale management, processing and mining of data sources to ever larger sizes and faster rates of transactions. One of the challenging applications is that of data streams where data sizes are so large that it is difficult to store all of the data in a database management system (DBMS) and there may be millions of "transactions" per second. We describe some quintessential examples as follows:

- Wireless sensor networks are ubiquitous [44, 118]. Sensor networks are used for location tracking [116], geophysical monitoring [118, 88], "communities" identification [108] thereof.

- There are multiple information and social networks among internet users, from web (e.g., blog) to chat networks (e.g., instant message (IM)). Given the trend in the expansion of information systems and the development of automatic data-collecting tools, data sets we will encounter in the future are expected to be more "massive" than what we have today.

- The volume of transaction log data generated is overwhelming. For example, there are approximately 3 billion telephone calls in U.S. every day, 30 billion emails daily. By comparison, the amount of data generated by world-wide credit card transactions is very small, at only 1 billion per month [94].

- In the networking community, there has been a great deal of interest in analyzing IP traffic data where the packets and flows ("connections") get

forwarded in high speed links [38]. Traffic monitoring ranges from the long term (e.g., monitoring link utilizations) to the ad-hoc (e.g., detecting network intrusions). Many of the applications operate over huge volumes of data (Gigabit and high-speed links), and have real-time reporting requirements.

In all above examples, data takes the form of continuous *data streams* rather than finite stored data sets, and long-running *continuous queries* are needed, as opposed to one-time queries. Traditional database systems and data processing algorithms are ill-equipped to handle complex and numerous continuous queries over data streams, and many aspects of data management and processing need to be reconsidered in their presence. Therefore, the research community has developed a large body of work in *data stream management systems* (DSMSs). Systems developed at various universities of this type include Aurora [4], STREAM [1] and TelegraphCQ [23], etc. Additional systems in this category are also developed in research labs like AT&T (Gigascope) [37], Sprint (CMON) [2], Telcordia [25]. Moreover, the academic system Aurora has given rise to the commercial StreamBase Systems Inc. [3] [1].

Prior work in data stream has shown how to estimate simple statistics such as distinct counts [51, 11, 34, 56, 58]), summary aggregates (e.g., heavy hitters [30, 24, 91], quantiles [63, 92, 30], histograms [65, 64, 60], wavelets [22]) and query results [30, 32, 7] (e.g., inner product, frequency moments). We need vastly more sophisticated statistical analyses in data stream models, which constitute the focus of this dissertation — statistical modeling of streaming data.

For statistical modeling of input data, we have three fundamental questions to address:

1. Model fitting — How to find parameters that make a model fit the data as

[1] http://www.streambase.com

closely as possible (given some assumptions)?

2. Model validation — How precise are the best-fit parameter values? Would another model be more appropriate?

3. Change detection — How to detect a change in models?

Model fitting and model validation are closely related. No matter there is a change in the underlying model or not, it is necessary for model validation to justify the correctness of the class of model chosen to begin with. If we start with a wrong model, even accurate model fitting is meaningless. Additionally, when modeling streaming data, the computational issue with regard to the three questions efficiently arises.

Generally speaking, developing a model of the underlying data source leads to a better understanding of the source and its characterization, which can lead to a number of applications in query processing and mining. For example, in sensor systems, models are found effective in offering noise resilient answers to a wide variety of queries (e.g., selectivity estimation [46], nearest neighbor queries [16, 100], and similarity searches [16], etc.) with much less network bandwidth usage by acquiring data only when it is not sufficient to approximate query answers with acceptable confidence [44, 42]. Models are also useful for trend analysis [103]. Once we have established a reliable model of "normal" behavior, then anomalies and trend changes, defined as deviations to the model, can be detected, we thus trigger actions or alarms.

We investigate two representative categories of modeling in data streams in this dissertation: parametric and structural modeling.

In statistics, a parametric model is a parametrized family of probability distributions, which describe the way a population is distributed. Assuming a model for the input stream, the goal of parametric modeling is to estimate its parameter(s) using one of the well-known methods such as regression fitting, maximum

likelihood estimation, expectation maximization and Bayesian approach on the data stream models. Due to the ubiquity of skew in data, as we can see in IP network [81, 113], financial [52], sensor [44], and text streams [32, 89] etc., we are particularly interested in modeling skew [2] in data streams. The challenges to do parametric model fitting at streaming speeds are both *technical* — how to continually find fast and reliable parameter estimates on high speed streams of skewed data using small space — and *conceptual* — how to validate the goodness-of-fit and stability of the model online. We show how to fit two models of skew — hierarchical (binomial multifractal) and non-hierarchical (Pareto) power-law models — on a data stream.

From another perspective, many real applications can be modeled using communication graphs where interested persons engage in various activities such as telephone calls, emails, Instant Messenger (IM), blog, web forum, e-business and so on. These arise in applications in characterizing an individual's communication behaviors, identifying repetitive fraudsters, message board aliases, extremists etc., using structural information described by graphs. Thus we call it *structural modeling*. Tracking such electronic identities on communication networks can be achieved if we have a reliable "signature" for nodes and activities. In particular, a signature for a node is a subset of graph nodes that are most relevant to the query node, such that the signature represents the query node's communication patterns. While many examples of signatures can be proposed for particular tasks [36, 70, 69], what we need is a systematic study of the principles behind the usage of signatures to any task. Our framework on signatures for communication graphs addresses this challenge and the scalability issues with respect to signature schemes.

[2] In particular, we are modeling skew in frequencies of domain values. So here "skew" means few item values are frequent, and we observe a long tail of infrequent values.

Another problem of interest, in parallel with modeling, in monitoring streams of data in a broad range of application is that of *change detection* [53, 72, 111]. A change in the data source can cause the models stale and degrade their accuracy, so a change detection system should respond promptly to a change when it happens. Parametric change detection methods [44, 42, 82] trigger an alarm when there is a significant change in parameter value(s). But real data often does not obey simple parametric distributions in practice. Therefore we need a non-parametric technique that makes no assumptions on the form of the distribution as *a priori*. To the best of our knowledge, prior approaches in database research to detecting a change in non-parametric distributions affix two windows to streaming data and estimate the change in distribution between them [39, 80]. Fixing a window size is problematic when the information on the time-scale of change is unavailable, and it is infeasible to exhaustively search over all possible window sizes either. We improve existing work by adapting the statistically sound *sequential probability ratio test* [109] to the online streaming case for change detection, such that the most likely change-point estimate is integrated into the test statistics, then changes at different scales can be detected without instantiating windows of different sizes in paralle.

Since the IP traffic analysis case is the most developed application for data streaming, we focus on network traffic data for application study of our proposed techniques. The contributions of this dissertation are as follows:

- *Parametric modeling.* (Chapter 3) We show how to fit hierarchical (binomial multifractal) and non-hierarchical (Pareto) power-law models on a data stream. We address the technical challenges using an approach that maintains a sketch of the data stream and fits least-squares straight lines; it yields algorithms that are fast, space-efficient, and provide approximations of parameter value estimates with *a priori* quality guarantees relative to those obtained offline. We address the conceptual challenge by designing

fast methods for online goodness-of-fit measurements on a data stream; we adapt the statistical testing technique of examining the quantile-quantile (q-q) plot, to perform online model validation at streaming speeds.

We complement our analytic and algorithmic results with experiments on IP traffic streams in AT&T's Gigascope® data stream management system, to demonstrate practicality of our methods at line speeds. We measured the stability and robustness of these models over weeks of operational packet data in an IP network. In addition, we study an intrusion detection application, and demonstrate the potential of online parametric modeling.

- *Structural modeling.* (Chapter 4) We develop a formal framework for the use of signatures on communication graphs and identify three fundamental properties that are natural to signature schemes: persistence, uniqueness and robustness. We justify these properties by showing how they impact a set of applications. We then explore several signature schemes — previously defined or new — in our framework and evaluate them on real data in terms of these properties. This provides insights into suitable signature schemes for desired applications. As case studies, we focus on the concrete application of enterprise network traffic. We apply signature schemes to two real problems, and show their effectiveness. Finally, we discuss scalability issues with respect to signature schemes.

- *Change detection.* (Chapter 5) We adopt the sound statistical method of sequential hypothesis testing to study the problem of change detection on streams, without independence assumption. It yields algorithms that are fast, space-efficient, and are oblivious to data's underlying distributions. Additionally, we formulate and extend our methodology to local change detection problems that have not been addressed earlier. We perform a thorough study of these methods in practice with synthetic and real data

sets to not only determine the existence of a change, but also the point where the change is initiated, with only a small delay between the change point and the ponit when it is detected. Our methods work seamlessly without window limitations inherent in prior work, and are highly effective at detecting changes quickly as our experiments show.

Chapter 2
Preliminaries

2.1 Computational Model

2.1.1 Massive Data Streams

A data stream is a sequence of data elements $x_1, x_2, \ldots, x_n$ from a data source, with $x_i \in U = \{0 \ldots u-1\}$. The semantics of the data element x_i may be different from application to application. This leads to different models of data streams. Each data element x_i may represent a number such as, an IP address, a telephone number, a student ID, and the ID of an item in transaction, etc. It may also represent an instance of interaction or communication between individuals; that is, $x_i = (s_i, d_i)$ denotes an interaction between s_i and d_i. Moreover, x_i can be attached with more sophisticated data semantics; for example, each x_i may represent a subset of the items that customers put in their market baskets for one transaction. With such a data stream, the input is also time series, where the ith entry in the sequence is a measurement at time i. In this dissertation, when referring to streams, we mean a data stream like the above one, and we do not consider streams where data might arrive out of (time) order.

There is another type of data stream [96], in which an implicit array S of domain size u is involved. Here $S[i]$ is the value of the ith entry of array S, and denotes the total for item $i \in U$ in the stream. Data elements in the stream may take the form (i, k), where $i \in U$ and $k \in \mathbb{Z}$. The semantics of such an element state that $S[i] = S'[i] + k$, where $S[i]$ and $S'[i]$ are respective value of that entry

after and before seeing the element. This leads to streaming models such as the *cash-register model* and the *Turnstile model* [96]. The difference between the two models is that, in the cash-register model $k \geq 0$; while in the Turnstile model, k could be negative.

All streaming problems considered in this dissertation assume that data elements are only added to the current set, but not deleted. This is because models with the full dynamic property are orthogonal to our proposed methodologies, so we do not consider those variants for now.

2.1.2 Window Models

A good model for streaming data should fit well on many subsets (window of data) to be considered robust. Due to the evolution of the real data, it is inaccurate to build models over the entire data stream. It is natural to imagine that the recent past in a data stream is more significant than the distant past. There are a variety of streaming windows to emphasize the data from the recent past:

- **Landmark windows** [55, 119] identify certain timepoints called landmarks in the data stream, and the aggregate value at a point is defined with respect to the data elements from the immediately-proceeding landmark k until the current point. A cumulative window is a special case when $k = 1$; in this case, all the available data are considered. An example of a landmark is when a user requests the computation of an *ad hoc* query.

- **Sliding windows** [40] are typically of a fixed size. One specifies a window size w, and explicitly focuses only on the most recent stream of size w. That is, at time n, only consider a sliding window of updates $x_{n-w+1}, \ldots, x_n$. Elements outside this window fall out of consideration for analysis, as the window slides over time.

- **Decaying window** [27] model believes that recent sliding windows are more important than previous ones, with weights of data from one window decreasing exponentially into the past. Formally, one considers the signal as fixed size windows of size w and the aging factor λ. Let B_i represent the signal from window i. We inductively maintain β_i as the meta-signal after seeing i windows. When the $(i+1)$th window is seen, we obtain

$$\beta_{i+1} = (1-\lambda)\beta_i + \lambda B_{i+1}.$$

  This weight function provides a smooth dynamic evolution of β_{i+1}. In addision, periodic updates do not require accessing transaction data for all previous time windows. All that is needed is β_i and the new set of transactions defined by B_{i+1}.

Ideally, we like methods to work on all of these window models so that we can flexibly try modeling the data in different ways.

2.1.3 Streaming Computational Model

A data stream contains large volume of data, but what makes data streams unique is the very large universe. For example, the universal size could be

- the number of distinct source, destination IP address pairs, which could be as large as 2^{64} in the IPv4 domain;
- the number of distinct http addresses on the web, which is potentially infinite since web queries get sometimes written into http headers;
- the size of a cross-product of the domains of the attributes of interest. This may lead to a potentially large domain space even if individual attribute domains are small.

A streaming algorithm is an algorithm that computes some function over a data stream at different times. These functions can be thought of as queries to the data structure updated during the stream. Due to the explosion of data, a desired streaming algorithm satisfies the following properties:

- **Storage.** All elements observed on the fly should be summarized in a synopsis structure [57] with space much smaller than the domain size u, typically polylogarithmic in it.
- **Per-item processing time.** Synopsis should be fast to update in order to match the streaming speeds. As is standard in dealing with data streams, each new element is handled in small time, typically polylogarithmic in u.
- **Query time.** Based on a synopsis of original data, functions on the input stream should be computed efficiently and preferrably with accuracy guarantees, provided *a priori*, so that queries can be evaluated on-demand frequently.
- **One pass, sequential access.** The input stream is accessed in a sequential fashion. The order of the data elements in the stream is not controlled by the algorithm. Moreover, it is preferrable for the algorithm to evaluate functions over the stream in one or at most a small number of passes.

These properties characterize the algorithm's behaviors during the time when it goes through the input data stream. The algorithm may perform certain pre-processing and/or post-processing on the workspace (not on the input stream) before and/or after this time.

Property 1 *[96] At any time n in the data stream, we would like the per-item processing time, storage as well as the computing time of a streaming algorithm to be simultaneously $o(u, n)$, preferrably, polylog(u, n), where u is the domain size.*

2.1.4 Semi-Streaming Computational Model

Consider the spectrum of the size of the storage space that an algorithm takes to access the input data. At one extreme of the spectrum, we have algorithms that can use memory large enough to hold the whole input. At the other extreme, we have streaming algorithms that use only polylogarithmic space. It has been suggested in [96] a middle ground:

Property 2 *[96] With the semi-streaming model, for a graph with n vertices, the algorithms use $O(n \cdot polylog(n))$ bits of space.*

That is, the space is proportional to n, the number of vertices, and sublinear in m, the number of edges. Some interesting results have been obtained in the semi-streaming model for graph properties such as diameters, spanners etc. [50, 49]. We consider this model in structural analysis of communication graphs for signatures (see Chapter 4).

2.2 Streaming Analysis Tools

Previous work on streaming algorithms has focused on computing statistics over the stream. In a data stream setting, with small-space data structures that can be updated for each input every efficiently, such algorithms cannot solve most problems on input exactly, so we will allow approximations. In fact, all algorithms we show will be probabilistic and will succeed with probability at least $1 - \delta$ and be accurate to some prespecified error ϵ, for parameters ϵ and δ. We discuss below the main mathematical and algorithmic techniques used in data stream models that serve as basic building blocks in solving our problems.

2.2.1 Random Projections

Random projection is a simple geometric technique for reducing the dimensionality, using projection along pseudo-random vectors. The pseudo-random vectors are generated by space-efficient computation of limitedly independent random variables. These projections are called the *sketches*. Many sketches have been proposed in DSMSs, each suitable of different sets of problems and with different time and space requirements [9, 96].

Count-Min (CM) Sketch [30, 32] is a small-space data structure that is useful for variety of approximations. It has similar performance as the best-known sketches [7, 24, 28, 74] in terms of accuracy for estimates we care about, but has the best update time.

The CM sketch with parameters (ϵ, δ) is a two dimensional array of counters with width l and depth $d = \lceil \ln \frac{1}{\delta} \rceil$: $count[1,1] \ldots count[d,l]$. Each entry of the array is initially zero. Additionally, d hash functions $h_1 \ldots h_d : U \rightarrow \{1 \ldots l\}$ are chosen uniformly at random from a pairwise-independent family. Given l and d, the space overhead is the ld counters and the d hash functions, each of which can be stored using two words. Whenever an update (i_t, c_t) arrives (e.g., tth packet with source i_t and bytesize c_t), c_t is added to only one count in each row, and the counter is determined by h_j. Formally, we set

$$\forall 1 \leq j \leq d : count[j, h_j(i_t)] \leftarrow count[j, h_j(i_t)] + c_t.$$

The update time depends on the depth d of the sketch. CM sketch can be used to answer a variety of queries. The output procedure varies with the application as well as the choice of sketch width l.

Fact 1 *With probability at least* $1 - \delta$*, CM-Sketch [30, 32] provides* ϵ *approximations to point, rangesum, inner-product queries,* L_2*-norm and quantile esitmations*

in small — typically $O(\frac{\log^2 |U|}{\epsilon^2} \log(\frac{\log |U|}{\delta}))$ *— space and per-item update time* [1].

Estimating number of distinct elements. This is a fundamental problem in data streams, and several approaches [11, 34, 51, 56, 58] have been proposed to approximate the number of distinct elements observed. We omit detailed discussions of these methods, and summarize the properties that they guarantee:

Fact 2 *There exist approximate distinct counter algorithms [51, 11] that take parameters* ϵ *and* δ *and create a data structure of size* $O(\frac{1}{\epsilon^2} \log 1/\delta)$ *machine words. The time to process each update is* $O(\log 1/\delta)$*. For an input stream of (integer) values, such algorithms report an estimate* $\hat{d}$ *for the distinct count such that, if* d *is the true number of distinct elements observed, then with probability at least* $1 - \delta$*,* $(1 - \epsilon)d \leq \hat{d} \leq (1 + \epsilon)d$*.*

2.2.2 Sampling Techniques

Sampling in the data stream context means every data element is seen but only a (polylogarithmic sized) subset of them are retained. Which subset of elements to keep can be chosen deterministically or in a randomized way. Various sampling algorithms have been proposed to estimate quantiles [63, 105], to find frequent elements [91], to estimate the inverse distribution [33], and to find rangesum of elements [6] in a data stream. But the worst case query cost of any sampling algorithm that (ϵ, δ)-approximates inner-product queries and L_2 norm — queries we care about — is more expensive than the cost for sketches with the same accuracy guarantees [10]. Therefore, we do not consider sampling algorithms in the rest of this dissertation.

[1] *Here, we have given a general upper bound that applies to the estimation of all the aggregates. In the remainder of the dissertation, we will provide bounds for various tasks that are specific to only the aggregates the tasks need.*

2.2.3 Other Algorithmic Techniques

Exponential Histograms (EHs) [40] are used to estimate all data counts in a recent window of size W. Datar et al provided tight bounds where $\log^2 n$ bits are sufficient and necessary for $(1+\epsilon)$ approximate estimates at time n. EH maintains buckets over ranges of data points. All data (i_t, c_t) seen in a time range $t \in (t_{i-1}, t_i]$ are aggregated into the ith bucket, and the EH maintains the value t_i and the count $C_i = \sum_{t_{i-1}<t\leq t_i} c_t$ for each bucket. Buckets where $t_i < n - W$ are discarded. When a new data point arrives, it is placed in its own new bucket. Buckets are then merged in a certain way such that there is always a logarithmic $O(\log n)$ number of buckets. At any given point n, the count estimate on the most recent W observations can be obtained from $\sum_i C_i$.

A characterization of the merging process of EH is that two consecutive buckets are merged if the combined count of the merged buckets is dominated by the total count of all more-recent buckets. A merger of more-recent buckets may lead to a cascade of at most $O(\log n)$ such mergers of less-recent buckets upon the arrival of a single new element. But the amortized number of mergers is $O(1)$ per new item. Furthermore, the sequence of bucket counts (from the most- to the least-recent) is a non-decreasing sequence of powers of 2, and for some $k = \Theta(1/\epsilon)$ and some $P \leq \log \frac{2n}{k} + 1$, for each possible count $2^p < 2^P$, there are exactly k or $k+1$ buckets having count 2^p, there are at most $k+1$ buckets having count 2^P, and no buckets have counts greater than 2^P.

Union Bounds [95]. This is very frequently used in streaming analysis. It derives directly from the inclusion-exclusion principle and states that the probability of the union of a set of events is at most the sum of the probabilities of the events. Formally, let $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$ be an arbitrary set of events, the bound says:

$$\Pr(\cup_{i=1}^{n} \mathcal{E}_i) = \sum_i \Pr(\mathcal{E}_i) - \sum_{i<j} \Pr(\mathcal{E}_i \cap \mathcal{E}_j) + \cdots \leq \sum_i \Pr(\mathcal{E}_i).$$

An important feature of the union bound is that there is no other condition on the

events for the bound to hold. In particular, when the events are not independent, the bound still holds. This is quite useful in places where independence among random variables cannot be achieved.

Chapter 3
Modeling Skew in Data Streams

3.1 Introduction

While much of prior work on data stream analysis has focused on nonparametric techniques such as approximate aggregates (e.g., heavy hitters, distinct counts), summaries (e.g., quantiles, histograms, wavelets) and query results (e.g., join sizes)—see [9, 96] for surveys—little effort has been focused on *modeling* the data stream parametrically. In general, developing a model of the underlying data source leads to a deeper structural insight into the source and its characterization, which can lead to a number of applications in query processing and data mining. For example, parametric models are used in selectivity estimation [46], trend analysis [103], outlier detection [101], and improving the quality guarantees of existing algorithms [32, 87]. Models are also useful for generating synthetic traces of data sources for experimental or simulation studies [110], model-driven data acquisition in sensor networks [43] and provisioning [113].

What are suitable models for data streams? There are suitable hierarchical as well as non-hierarchical models.

Data streams have hierarchical dimensions, such as IP addresses in network data, locations in sensor data, and time indexed share volumes in financial streams. Models for hierarchical data have focused on multi-scale properties over different levels of aggregation. Recently, in the networking community, the IP address space has been shown to be well modeled using the *fractal dimension* [81]. Using a variety of traces of network traffic, the authors considered the fractal dimension

of the traffic over the IP address space, and showed it is (a) stable over short time intervals at local sites; (b) different at different sites with different characteristics; and (c) different with changes in traffic conditions such as during a worm attack. Hence, it is possible that the fractal dimension of the IP address distribution can serve as a "fingerprint" for normal traffic, and that sudden deviations from this would signal an anomaly or a change in network trends.

At a non-hierarchical view, streaming data is also teeming with *high variability*—most observations take small values, while a few observations attain extremely large values with non-negligible probabilities. This phenomena is ubiquitous in internet traffic—IP flow sizes, TCP connection duration, request interarrival times, node degrees of inferred Autonomous System, and file sizes transferred over the Internet; see [112] for references. All these are well modeled by heavy-tailed—in particular, Pareto—distributions and are characterized by (non-hierarchical) skew. Moreover, Pareto/power-law modeling of financial streams of stock prices and number of trades is useful for real-time forecasting and options pricing [52].

What are the revelant issues in fitting models to streaming data? Typically, model fitting is done offline when it applies to stable situations where it is done once or infrequently, such as in [81, 112]. However, the dynamics of streaming data calls for continuous parameter estimation and validation of models to find reliable models that are robust over long time intervals. For example, at any time an unnoticed DDoS attack may be altering the distribution of IP address prefixes in the traffic; this is not a rare occurrence in actual measurement studies [83]. This suggests the need for a higher threshold for accepting a model, to reduce its susceptibility to outliers by demonstrating temporal stability. In a data stream context such as IP traffic modeling, it is impractical to gather data frequently and perform offline analyses. The preferred approach is to perform estimation on the data stream directly as the data is generated. It is therefore imperative that estimation work at line speeds (i.e., rates at which packets are forwarded in the

links). As a consequence, there are two outstanding concerns in modeling data streams:

1. How to estimate the model parameter(s) on data streams? Well known statistical tools such as EM [62] are both computationally expensive and require a large amount of storage to provide accurate estimates, and are therefore infeasible within the space and time constraints of a Data Stream Management System (DSMS).

2. How to validate the goodness-of-fit of the model with the estimated parameter(s) on data streams? Using traditional statistical hypothesis testing techniques is typically infeasible within the space and time constraints of a DSMS.

These concerns are broadly applicable to all modeling applications. We address these concerns and present methods for hierarchical and non-hierarchical modeling in data streams. Our contributions are as follows:

- To the best of our knowledge, this is the first work on model fitting over data streams *with* a priori *error guarantees*. We show how to maintain an estimate of the binomial multifractal ("b-model") and Pareto model parameters using few memory updates per item and small space; our algorithm *provably* output parameter estimates to within an *additive* error bound that can be prespecified.

- We present highly efficient streaming methods to validate the model parameters online using quantile-quantile ("q-q") plots from statistics, which are widely used for power-law fitting. This requires computing order statistics of the ideal models: we propose methods to compute them without actually generating data, which would be prohibitive on a data stream.

- We complement our analytical and algorithmic results with a detailed experimental study on a variety of real IP traffic data streams from AT&T. In particular, we have implemented our methods within the Gigascope system that is operational inside the AT&T network, to perform model fitting at speeds of 100K packets per second using only 2% of CPU utilization. We ran this experiment over a period of several weeks to measure the robustness and stability of the model. We also perform a detailed study of applicability of modeling to an application—intrusion detection—with labeled data. These real-data experiments provide many insights into the fit of models on packet-level data and their applicability, something that could not have been possible without our proposed streaming solutions. As our experience shows, prior offline trace-driven study provides only limited insight into the applicability of model that fit IP streams, and careful interpretation of the evolution of the model parameters as well as the quality of fit of the model are needed to reason about streaming data.

The chapter is organized as follows. We present a high level view of our approach in Section 3.2. We present our algorithmic solutions for modeling and validating (hierarchical) b-model in Section 3.3 and present experimental studies in Section 3.5. Likewise, we present algorithmic methods and experimental studies in Sections 3.4 and 3.6 respectively for (non-hierarchical) Pareto model fitting and validation. Our overall approach can be extended to other models and streaming applications quite naturally. Section 3.7 describes these extensions. Related work is in Section 3.8 and conclusions are in Section 3.9.

3.2 Model Fitting on Data Streams

Let $U = \{0, ..., |U|-1\}$ be the data domain. Our goal is to model the distribution $S[0, \ldots, |U|-1](t)$ where $S[i](t)$ is the *frequency* of item i after seeing the t-th

input. Say the $(t+1)$-th input is item i. Then $S[j](t+1) = S[j](t) + 1$ for $j = i$, and $S[j](t+1) = S[j](t)$ otherwise. (We omit the timestamp t hereafter.) So each new input is an update to S. For example, for the IP domain, $0, ..., |U|-1 = 2^{32}-1$ could be the source IP addresses, $S[i]$ could count the number of packets sent by i, and each new packet on the network link is an update. As is standard by now in dealing with such streams, we will design methods that use space much smaller than $|U|$, typically polylogarithmic in it; likewise, each new item is handled in small time, typically polylogarithmic in $|U|$ [96].

Fitting models to a stream S consists of two tightly coupled problems: *parameter estimation* and *model validation*. A large variety of techniques exist for obtaining parameter estimates from a sample of data; we discuss their adaptation to data streams and our methods in Sections 3.3 and 3.4. Model validation involves comparing the model (instantiated with the estimated parameter(s) obtained from using one of the available techniques), with the actual data set by performing a *goodness-of-fit* test. This is necessary because the class of model chosen may be wrong to begin with. In this case, even accurate parameter estimates would be bogus.

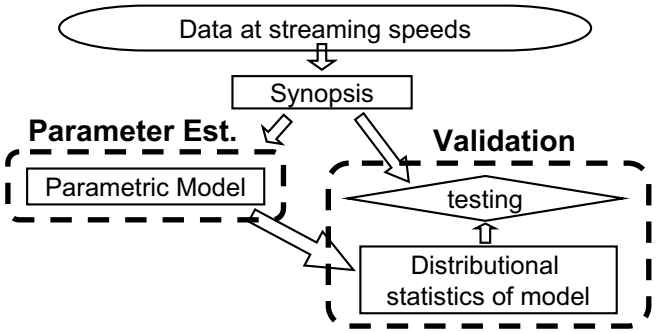


Figure 3.1: Overview of a streaming algorithm for model fitting.

The overall process of our parameter estimation and validation is shown in Figure 3.1. Our challenge is to perform all tasks on high-speed data streams. A suitable window (e.g. a *landmark window* [55] which is a window of fixed length) of the stream is summarized using a synopsis data structure. This is used for

model fitting which relies on estimating input stream statistics (eg. L_2 norm estimation). Simultaneously, model validation is done and this too uses different statistics on the input stream (eg., quantiles, rangesum queries). Further, model validation requires statistics of the model-generated data without explicitly generating the data which would be prohibitive. By carefully designing the methods, we are able to use a single synopsis data structure—CM Sketch [30]—to supply all the statistics we need for model fitting and validation, as summarized in Fact 1.

3.3 Hierarchical (Fractal) Model

Much of stream data is hierarchical, such as IP addresses in network data, locations in sensor data, and transaction rates in financial time series data. Models for hierarchical data have focused on multi-scale properties such as self-similarity over different levels of aggregation, widely known as a (recursive) '80-20' Law [47]. Here, we study its general version called binomial multifractal, or the b-model.

3.3.1 Model Definition

A binomial multifractal b-model can be constructed through a *multiplicative cascade* [90, 61, 104, 47, 110], with bias $b \in [0.5, 1]$ as the only model parameter. The process starts with an initial mass M distributed over attribute domain U. The first stage of the cascade construction divides U into *dyadic* intervals $[0, |U|/2 - 1]$ and $[|U|/2, |U| - 1]$; and assigns mass $(1 - b)M$ to the lower half domain, bM to the upper half. Iterating this construction process, we recursively divide each parent interval into its two dyadic subintervals, and assign $1 - b$ and b fraction of the parent mass to the left and right subinterval. Formally, mass assignment is as follows: $M_0^{(0)} = M$, $M_{2k}^{(p+1)} = (1 - b)M_k^{(p)}$ and $M_{2k+1}^{(p+1)} = bM_k^{(p)}$, where $M_k^{(p)} = \sum_{i \in U_k^{(p)}} S[i]$ (called a *p-aggregate*) indicates the mass associated with dyadic interval $U_k^{(p)} = (k2^{\log|U|-p}, (k+1)2^{\log|U|-p} - 1)$, $p = 0, \ldots, \log|U|$

and $k \in \{0, \ldots, 2^p - 1\}$. Note that $U_k^{(p)}$ refers to the set of attribute values sharing the prefix with length p—a.k.a. a *p-bit prefix*, valued k. Figure 3.2(a) shows an example. The structure of the b-model is a binary tree. The model,

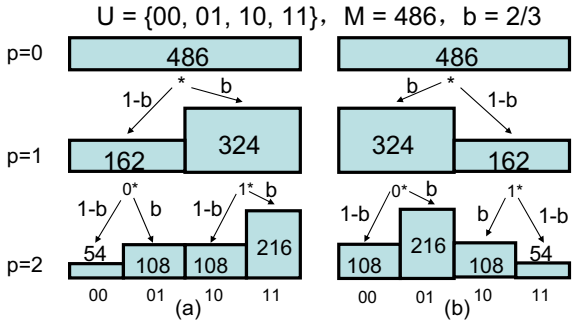


Figure 3.2: Two examples of a b-model ($b = \frac{2}{3}$) where both data sets have the same b-value. In (a), $M_0^{(0)} = 486, M_0^{(1)} = 162, M_1^{(1)} = 324, M_0^{(2)} = 54, M_1^{(2)} = M_2^{(2)} = 108, M_3^{(2)} = 216$.

as described thus far, is deterministic, but more generally, b can go arbitrarily to the left or to the right. Note that multiple data sets can map to the same b value; Figure 3.2(b) gives a different data set with the same b-value as that in Figure 3.2(a). Due to the multiplicative cascade process being applied at every level, the model exhibits self-similarity—parts of the data are similar (exactly or statistically) to the whole—over all scales. Hence, the b-model is a multi-fractal model.

3.3.2 Fractal Parameter Estimation

The method that is commonly employed to estimate b in the b-model is as follows (see [61, 110, 104]). We will first describe this method, and our contribution will be to adapt it to the online streaming case and provide approximation guarantees.

First, a related parameter called the *correlation fractal dimension* D_2 is estimated by line-fitting. Let $N(p) = \sum_{i=0}^{2^p-1} (M_i^{(p)})^2$; then the *correlation fractal dimension* D_2 is defined as (see [104]):

$$D_2 \equiv \lim_{p \to \infty} \log N(p)/p.$$

From the definition it follows that the plot of $(p, \log N(p))$ is a straight line, the slope of which is D_2. Figure 3.3 gives an example that shows D_2 of the above data sets. In real data sets, seldom does one see such a perfect straight line.

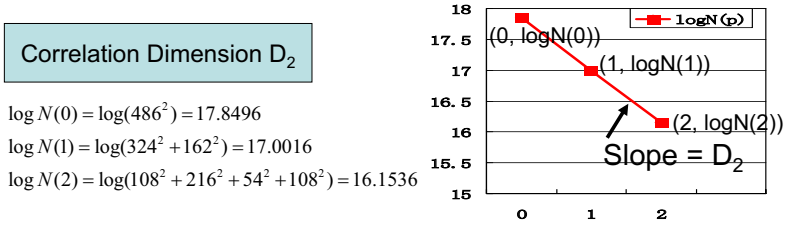


Figure 3.3: Computing D_2 for the example in Figure 3.2.

In practice, D_2 is calculated by plotting the points $(p, N(p))$ for various p and fitting the best least-squares fit line to the points. Formally, for the point set $P = \{(x_p, y_p) \mid p \in V\}$, slope of the least-squares fit line (that minimizes the least squared error) is

$$D_2 = \frac{\sum_{p=0}^{\log|U|}(x_p - \bar{x})y_p}{\sum_{p=0}^{\log|U|}(x_p - \bar{x})^2} = \sum_{p=0}^{\log|U|} k_p y_p,$$

where $k_p = \frac{x_p - \bar{x}}{\sum_{p=0}^{\log|U|}(x_p - \bar{x})^2}$, and $\bar{x}$ is the average of $\{x_p \mid p \in V\}$. Then, this value D_2 is used to derive an estimate for b analytically. It is known that $D_2 = \log(2b^2 - 2b + 1)$ [61]. Inverting this formula, we can get b for a given value of D_2. Sort-based [16] and hash-based [78] algorithms have been proposed for this, but they require storing the entire data set and are thus infeasible on a data stream.

Proposed Streaming Method. We first obtain an estimate $\hat{D}_2$ for the fractal dimension, via line fitting on log-log scales, and then extract parameter estimate $\hat{b}$ from $\hat{D}_2$ algebraically.

Set $V = \{0, \ldots, \log|U|\}$. The method for estimating D_2 involves $N(p)$ for all p's, where $N(p)$ denotes the sum of squares over all p-aggregates, $p \in V$. We use the CM-Sketch to do this. Then, $\forall p$, the estimate $\hat{N}(p)$ satisfies the (ϵ, δ) guarantee as in Fact 1. Given $F = \{(x_p = p, y_p = \log N(p)) \mid p \in V\}$

and $\hat{F} = \{(x_p = p, \hat{y}_p = \log \hat{N}(p)) \mid p \in V\}$, the exact and approximate fractal dimensions D_2 and $\hat{D}_2$ are respectively computed by the slope of the least-squares fit lines to F and $\hat{F}$.

Theorem 1 *Given with probability at least* $1 - \frac{\delta}{\log |U|}$, $(1-\epsilon)N(p) \leq \hat{N}(p) \leq (1+\epsilon)N(p)$, $\forall p \in V$, *the estimate* $\hat{D}_2$ *with probability at least* $1-\delta$ *satisfies*

$$|\hat{D}_2 - D_2| \leq \frac{3\epsilon}{(1+\log |U|)(1-\epsilon)}.$$

Proof. Note that every x-coordinate of $\hat{F}$ is accurate and the y values are approximate. $\forall p \in V$, with probability at least $1 - \frac{\delta}{\log |U|}$, we have

$$\log(1-\epsilon) \leq \log \hat{N}(p) - \log N(p) \leq \log(1+\epsilon).$$

Let $A = \{p \mid k_p \geq 0, p \in V\}$ and $B = V - A$. Since $\sum_{p \in V} k_p = 0$, $\sum_{p \in A} k_p = -\sum_{p \in B} k_p = c$. Thus with probability at least $1-\delta$, [1] we have

$$\begin{aligned} c\log(1-\epsilon) &\leq \sum_{p \in A} k_p (\log \hat{N}(p) - \log N(p)) \leq c \log(1+\epsilon) \\ -c\log(1+\epsilon) &\leq \sum_{p \in B} k_p (\log \hat{N}(p) - \log N(p)) \leq -c \log(1-\epsilon) \end{aligned}$$

Summing up the terms, we get

$$|\sum_{p \in V} k_p (\log \hat{N}(p) - \log N(p))| \leq c \log(1 + \frac{2\epsilon}{1-\epsilon}).$$

We have $\log(1 + \frac{2\epsilon}{1-\epsilon}) \leq \frac{2\epsilon}{1-\epsilon}$. Thus $|\hat{D}_2 - D_2| \leq c\frac{2\epsilon}{1-\epsilon}$. From the definition of k_p,

$$c = \sum_{p \in A} k_p = \frac{\sum_{p=\log |U|/2}^{\log |U|} (p - \log |U|/2)}{\sum_{p=0}^{\log |U|} (p - \log |U|/2)^2} = \frac{\sum_{p=0}^{\log |U|/2} p}{2\sum_{p=0}^{\log |U|/2} p^2} = \frac{3}{2(1+\log |U|)}.$$

Thus $|\hat{D}_2 - D_2| \leq \frac{3\epsilon}{(1+\log |U|)(1-\epsilon)}$. ∎

[1] For the $\log |U|$ estimations for $N(p), p \in \{1, \ldots, \log |U|\}$, the probability of failure for each is $\frac{\delta}{\log |U|}$. Then applying the union bound ensures that, over all the queries, the total probability that any one (or more) of them overestimated by more than a fraction ϵ is bounded by δ, and so the probability that every query succeeds is $1-\delta$.

Discussion. An important contribution of our work is the theorem above that gives a strong guarantee about estimation error of D_2. In particular, the error does not increase with D_2 like multiplicative errors, and as $|U| \to \infty$, the error goes to 0, which is unusual for streaming algorithms. Finally, and this is crucial, the error is guaranteed no matter how the error in estimating $N(p)$ is distributed. In particular, we do not make any assumption about the errors in $N(p)$ estimation are, say, uniformly random. Using the CM-Sketch or any other sketch, typically does not guarantee that the estimation errors for $N(p)$'s are random. Fortunately, we do not need this property. The proof above shows that even adversarial errors in estimating each $N(p)$'s (within the specified error bounds) cannot affect our estimation of D_2 significantly.

The kernel of the proof shows that given a set $\{(x_i, \log y_i) \mid i \in V\}$ of points, if the approximation error of $\hat{y}_i$ for y_i is multiplicative and each x_i is exact, then the estimate for the slope of the least-squares fit line has an additive error that does not depend on its true value. This is a general claim thatis true beyond its applicability to fractal parameter estimation above. We will extend this analytic methodology later to other models. ∎

Now we focus on estimating b from $\hat{D}_2$. Recall that $D_2 = \log(2b^2 - 2b + 1)$; hence, inverting, $b = \frac{(1+\sqrt{2^{D_2+1}-1})}{2}$. In the following, let $\hat{b}_2$ represent the b estimate derived from estimated fractal dimension $\hat{D}_2$. We replace ϵ in Theorem 1 with $\tilde{\epsilon}$, set $\epsilon = \frac{3\tilde{\epsilon}}{(1+\log|U|)(1-\tilde{\epsilon})}$, and get the accuracy guarantees on $\hat{b}_2$.

Theorem 2 *Our algorithm uses* $O(\frac{\log|U|}{\epsilon^2}\log\frac{\log|U|}{\delta})$ *space,* $O(\log\frac{\log|U|}{\delta}\log|U|)$ *per-item processing time, and with probability at least* $1-\delta$*, outputs estimate* $\hat{b}_2$ *for* b *in* $O(\frac{1}{\epsilon^2}\log|U|\log\frac{\log|U|}{\delta})$ *time, such that:*

$$2^{-\frac{\epsilon}{2}}b_2 - 2^{-\frac{\epsilon}{2}-1}(\sqrt{2^{\epsilon}-1}+1) + \frac{1}{2} \leq \hat{b}_2 \leq 2^{\frac{\epsilon}{2}}b_2 + 2^{\frac{\epsilon}{2}-1}(\sqrt{1-2^{-\epsilon}}-1) + \frac{1}{2}$$

Proof. We show the upper bound on accuracy; the lower bound is similar and the

space and times follow from the use of CM-Sketch with appropriate parameters.

$$\begin{aligned}
\hat{b}_2 &= \frac{1+\sqrt{2^{\hat{D}_2+1}-1}}{2} \\
&\leq \frac{1+\sqrt{2^{D_2+\epsilon+1}-1}}{2} = \frac{1}{2} + 2^{\frac{\epsilon}{2}}\sqrt{2^{D_2-1} - \frac{1}{4}2^{-\epsilon}} \\
&\leq \frac{1}{2} + 2^{\frac{\epsilon}{2}}(\sqrt{2^{D_2-1} - \frac{1}{4}} + \frac{1}{2}\sqrt{1-2^{-\epsilon}}) \\
&= 2^{\frac{\epsilon}{2}} b_2 + 2^{\frac{\epsilon}{2}-1}(\sqrt{1-2^{-\epsilon}} - 1) + \frac{1}{2}
\end{aligned}$$

∎

To give some intuition, when $\epsilon = 0.01$, we have $0.99654 b_2 - 0.027629 \leq \hat{b}_2 \leq 1.00347 b_2 + 0.027776$.

3.3.3 Model Validation

Goodness-of-fit hypothesis tests for validating estimated model parameter(s) compare distributional statistics of the actual data against that generated by the model with the estimated parameter value(s). A standard test uses quantile-quantile (q-q) plots. The q-q plot [45] graphs the respective quantiles X and Y of real and generated data sets; if the two data sets are from the same distribution, then the respective quantiles should be roughly equal. In practice, the *correlation coefficient*, denoted cc, of the quantile pairs $(X, Y) = \{(x_i, y_i) | i = 1 \ldots N\}$ from the respective distributions is used to indicate goodness-of-fit based on such plots (e.g., see [18]).

$$cc = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}},$$

where, as usual, $\bar{x}$ is the mean of the x_i's, $\bar{y}$ is the mean of y_i's. We use this method.

We can get approximate ϕ-quantiles of the input data using the CM-Sketch [30]. However, finding quantiles of generated data using the b-model is a potential bottleneck on a stream. A naive approach is to actually generate such data and then

compute its quantiles, but this is infeasible on a data stream because the cardinality of generated data should be comparable to size of the input stream for accurate validation. Instead, we show how to find these quantiles *without materializing* such data, which allows for fast q-q testing in small space.

Online Quantile Generation. Our algorithm for generating quantiles directly from a b-model has a similar flavor to the stack-based algorithm in [110], which generates data according to parameter b. But whereas their algorithm visits *every* tree node in depth-first order (from left to right) and randomly flips a coin at each node to assign weights b and $(1-b)$ to the node's two children, our algorithm differs in two ways. First, since we are comparing the fractal model against a specific data set, instead of assigning weights randomly, they are assigned *deterministically* to match the input data; point queries to the CM-Sketch are used for this, by computing the corresponding prefix aggregates from the input and assigning b to the larger aggregate and $(1-b)$ to the smaller. This is much less likely to reject a valid model than randomly assigning the weights. Second, tree nodes which do not contain any ϕ-quantiles in their subtree are pruned. To do this, the algorithm maintains a cumulative sum, *percent*, of the weights up to current node. If $\lceil percent/\phi \rceil = i$ before visiting a node, then this node's entire subtree can be eliminated if it will not increase $\lceil percent/\phi \rceil$ to at least $i+1$. See Algorithm 1, where we make use of the following CM Sketch operator:

- CMH_Count(*cmh*, *level*, *index*) returns the estimate of the (*index*)-th *level*-aggregate, which is used to deterministically assign b to the larger aggregate.

Example. Consider the b-model representation in Figure 3.4. Let $b = 2/3$ and an original data distribution S on domain $U = \{0, 1, 2, 3\}$. $S[i]$ represents the number of times we observe value i from the input data, $S[i \ldots j] = \sum_{k=i}^{j} S[k]$,

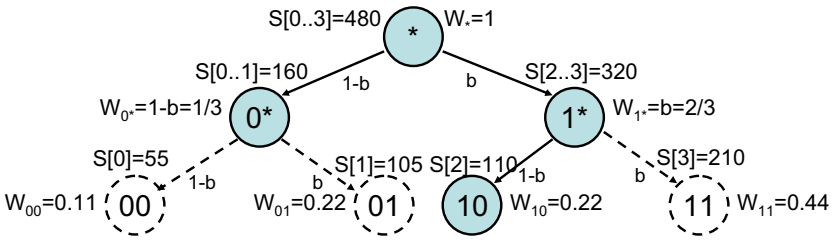


Figure 3.4: An example of the quantile generation algorithm.

and w_i is the weight of the subtree rooted at node i with regard to the fractal model with parameter value b. That is, $w_i = b^m(1-b)^{(p-m)}$, where m is the number of edges weighted b along the path from the root to node i, and p is node i's aggregation level. We are looking for the median ($\phi = 0.5$) of generated data that best fits the input data without data materialization. Before visiting the root node, $percent = 0$, since $percent + w_* = 1 > \phi$, current tree must contain a median. We assign weight $1-b$ to its left child and b to the right since prefix aggregate $S[0\ldots1] < S[2\ldots3]$. Then we prune the subtree rooted at $0*$ because $percent + w_{0*} < \phi$, which means the median does not appear inside its subdomain. When the next node $1*$ is visited, *percent* is increased to $1/3$. Similarly, we can easily tell that median is within the subdomain rooted at $1*$. Again b and $1-b$ are assigned according to prefix aggregates. And when a leaf node 10 is reached, because $percent + w_{10} > \phi$, we return $(10)_2 = 2$ as median of the generated data. In Figure 3.4, nodes in grey are traversed and nodes with dotted circles are pruned.

Theorem 3 *Our algorithm takes* $O(\frac{\log^2|U|}{\epsilon}\log\frac{\log|U|}{\delta})$ *space and* $O(\frac{\log^2|U|}{\phi}\log\frac{\log|U|}{\delta})$ *time to generate approximate* ϕ*-quantiles directly from a b-model.*

Proof. Only nodes whose subtree contains at least one quantile are visited. Since there are $O(1/\phi)$ quantiles in all, only that many nodes per level will be visited during the quantile search process. The space and time complexity follows from those of the CM-Sketch used for $O((1/\phi)\log|U|)$ range sum queries. ∎

Thus, the entire q-q test validation process can be run on a data stream with

small space and fast update.

3.4 Non-Hierarchical (Pareto) Model

IP streaming data such as IP flow sizes, TCP connection duration, request interarrival times, node degrees of inferred Autonomous System, or file sizes transferred over the Internet have high variability; these are modeled by the Pareto distribution.

3.4.1 Model Definition

Define a discrete random variable X with ordered domain $U = \{0, \ldots, |U| - 1\}$ to be an item observed in the input stream. Recall that $S[0 \ldots |U| - 1]$ represents the distribution we want to model where $S[i]$ is the frequency of item i. $\forall x \in U$, a discrete *Pareto Model* [5] with parameter z, $0 < z < 2$, called the *tail index*, satisfies $F(x) = \Pr(X > x) = \sum_{i=x+1}^{|U|-1} S[i]/N = (c/x)^z$, where N is the total number of observations, c is a scaling constant to ensure $\Pr()$ a probability function. Note that $F(x)$ denotes the *complementary* cumulative distribution function (CCDF).

3.4.2 Pareto Parameter Estimation

When plotted on log-log scales, $(x, F(x))$ appears (ideally) as a straight line with slope $-z$; this is the basis of commonly employed Pareto estimation methods [12, 102, 114, 112]. Our problem is to estimate z and validate the goodness-of-fit on a data stream.

Given a data set perfectly fitting a Pareto model with tail index z, $\forall x_1, x_2 \in U, x_1 \neq x_2$, the slope of the line penetrating points $(\log x_1, \log F(x_1))$ and $(\log x_2, \log F(x_2))$ is the desired $-z$. However, with real-life data, the CCDF on log-log scales is not likely to be a precise straight line. On one hand, deviations from the

line are usually observed at the head, known as *top concavity*. On the other hand, *sampling effects* occur at the end of the tail (higher slope than the true value) because there is a very small amount of data with extremely large values. See Figure 3.5 for a typical CCDF plot of a fitted Pareto (originally in [112]). Therefore, typically, the part of the distribution—the tail excluding extreme val-

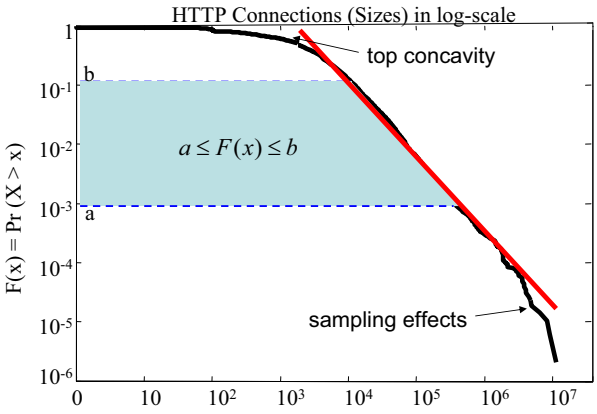


Figure 3.5: CCDF plot of fitted Pareto for HTTP connection sizes

ues at the ends—which looks most "Pareto-like" is used to fit the model [12, 102]. Formally, let $F(x) = \sum_{i \in U, i > x} S[i]/N$ where N is the total number of observed items, that is $\sum_i S[i]$. A regression line is fitted to the set of points $P_{a,b} = \{(\log(x), \log F(x)) | a \le F(x) \le b, x \in U\}$, for user-specified a and b where $0 < a < b < 1$. Again, this computation requires storing the entire data set, making it infeasible on a stream.

Proposed Streaming Method. We use the CM-sketch as before. However, a direct combination of CM-Sketch with the offline algorithm has a major drawback: we need to consider all x such that $a \le F(x) \le b$, which is time-consuming. We can prove much like in Theorem 1 that such an approach will give an accurate estimate for z in time $O(|U|)$ in the worst case. However, in order to get a faster estimation algorithm, we propose the use of CM-Sketch to pick out x_j's such that

x_j's are approximate ϕ-quantiles, $0 < \phi < 1$, and $a \leq F(x_j) \leq b$. Using CM-Sketch, we get x_j and the approximate value of $F(x_j)$, denoted $\hat{F}(x_j)$. Let P be the set of all $(\log x_j, \log \hat{F}(x_j))$ such pairs. We will find the least squares fit to these pairs and take its slope as the estimate $-\hat{z}$. We can present an estimate for the accuracy for this method by comparing against an offline algorithm. Suppose z is obtained by plotting the set of points Q that is the set of all $(\log x_j, \log F(x_j))$ pairs, because $F(x_j)$'s can be determined offline exactly. The slope of the best-fit line gives $-z$ for the Pareto parameter.

Theorem 4 *Our algorithm uses* $O(\frac{\log^2 |U|}{\epsilon a} \log \frac{\log |U|}{\delta \phi})$ *space,* $O(\log \frac{\log |U|}{\delta \phi} \log |U|)$ *per-item processing time, and with probability at least* $1 - \delta$*, outputs estimate* $\hat{z}$ *for* z *in* $O(\frac{\log |U|}{\phi} \log \frac{\log |U|}{\delta \phi})$ *time such that:*

$$|\hat{z} - z| \leq 2\epsilon/(1 - \epsilon).$$

Proof. With $O(\frac{\log^2 |U|}{\epsilon'} \log \frac{\log |U|}{\delta'})$ space and per-item update time of $O(\log \frac{\log |U|}{\delta'} \log |U|)$, CM Sketch returns $(x_j, \hat{F}(x_j))$ such that with probability at least $1 - \delta'$, $F(x_j) - \epsilon' \leq \hat{F}(x_j) \leq F(x_j) + \epsilon'$. Note that $\forall x_j, F(x_j) \geq a$, so

$$\begin{aligned} (1 - \epsilon'/a)F(x_j) &= F(x_j) - \epsilon' F(x_j)/a \leq \hat{F}(x_j) \\ &\leq F(x_j) + \epsilon' F(x_j)/a \\ &= (1 + \epsilon'/a)F(x_j). \end{aligned}$$

Set $\epsilon' = \epsilon a$, then $(1 - \epsilon)F(x_j) \leq \hat{F}(x_j) \leq (1 + \epsilon)F(x_j)$. Therefore, our defined set of points Q and P, from which z and $\hat{z}$ are computed by the slope of regression lines, correspond to respective point set F and $\hat{F}$ defined before Theorem 1. The rest of the proof of accuracy is similar to the proof of Theorem 1, and the time and space bounds follow by replacing ϵ' and δ' with ϵa and $\delta \phi$, respectively. The time to output $\hat{z}$ is the time to find $O(1/\phi)$ quantiles. ∎

3.4.3 Model Validation

Fractal model validation (see Section 3.3.3) relies on q-q plots, where the correlation coefficient (cc) is used as a heuristic to measure the goodness-of-fit. The Pareto model has infinite variance [112] and hence, the cc-based q-q test fails directly because correlation is not even defined unless variances are finite. As a result, both numerator and denomenator of the cc-formula are dominated by extremely large values, leading the ratio to 1 regardless of whether the corresponding z estimate is close to its true value or not. Therefore, we use q-q plot as a visual construct to evaluate the goodness-of-fit. The basic building blocks of a q-q plot are quantiles of real and generated data sets. Quantiles of real data can be approximated accurately and efficiently by using the CM-Sketch, as before.

Online Quantile Generation. Finding quantiles of generated data without materializng the entire data set is again a challenge. Luckily, the Pareto model has well-defined CCDF $\Pr(X > x) = (c/x)^z$, where $c = [z \sum_{x \in U} x^{-(z+1)}]^{-1/z}$. We can derive that its i-th ϕ-quantile $x_i = c(i\phi)^{-1/z}$. We do not know c even though we have the model parameter z. However, any c will do since the expression for ϕ-quantile says that the quantiles are linearly related under different c's. Hence, if quantiles of generated data are computed via any choice of c-value, the resulting q-q plot should still be a straight line, though its slope is not necessarily 1. Since the focus of our testing hypothesis is to validate the correctness of z, rather than c, we use some arbitrary c to generate online q-q plot, to test goodness-of-fit of the model, by comparing the generated plot against a straight line. Hence,

Theorem 5 *Our algorithm uses $O(1/\phi)$ time and space to generate approximate ϕ-quantiles directly from the Pareto model without materializing data.*

Proof. First, select some c-value. Then the i-th ϕ-quantile $x_i = c(i\phi)^{-1/z}$ is computed in $O(1)$ time and space. There are at most $1/\phi$ ϕ-quantiles to generate

from the region $[a, b]$ of interest. ∎

3.5 Fractal Model Fitting Experiments

In this section we evaluate the accuracy and performance of our proposed streaming methods for fitting a b-model (see Section 3.3) against alternative methods. Also, we implemented our method in AT&T's Gigascope data stream management system to demonstrate its practicality on a live data stream and examine the robustness and stability of b-model fitting on several weeks of operational data in an IP network. Finally, we study an intrusion detection application using the b-model on traffic data labeled with known attacks.

3.5.1 Alternative Streaming Methods

We consider two alternative methods for estimating b: one based on Maximum Likelihood Estimation (MLE), a common approach to model fitting in statistics; and a straightforward heuristic which we dub *pyramid approximation*. We will later compare these against our proposed method from Section 3.3, both in concept and in practical performance.

Method 1: Maximum Likelihood Estimation.

Maximum likelihood estimation (MLE) of the parameter b involves finding the value that maximizes the *a posteriori* likelihood of occurrence based on the existing data. MLE is a time-honored technique described in many classical statistics textbooks. Consider a single stage of the cascade construction with N items observed in the parent interval, N_1 in the heavier subinterval weighted b, and $N - N_1$ in the other. The probability of observing such an item distribution given b is $b^{N_1}(1-b)^{N-N_1}$. We can use this for multiple levels to derive formulas for b. Define event $A = \{\text{number of items observed at } U_i^{(k)} = M_i^{(k)}, \forall i \in \{0, \ldots, 2^k - 1\}\}$, where $M_i^{(k)}$ is the ith k-aggregate in $U_i^{(k)}$ at level k. Define M to be the total item

count, and $B^{(k)} = \{i \mid U_i^{(k)}$ is assigned b fraction of the parent $(p-1)$-aggregate and $i \in \{0, \ldots, 2^k - 1\}\}$. Then the likelihood function is

$$\begin{aligned} L_p(b|data) = \Pr[A|data] &= \prod_{k=1}^{p} \prod_{i \in B^{(k)}} b^{M_i^{(k)}} (1-b)^{M_{\lfloor i/2 \rfloor}^{(k-1)} - M_i^{(k)}} \\ &= b^{\sum_{k=1}^{p} \sum_{i \in B^{(k)}} M_i^{(k)}} (1-b)^{pM - \sum_{k=1}^{p} \sum_{i \in B^{(k)}} M_i^{(k)}}. \end{aligned}$$

The log-likelihood function is

$$\log L_p(b|data) = (\sum_{k=1}^{p} \sum_{i \in B^{(k)}} M_i^{(k)}) \log b + (pM - \sum_{k=1}^{p} \sum_{i \in B^{(k)}} M_i^{(k)}) \log(1-b).$$

Setting the derivative with respect to b to zero, we have $\hat{b}_{MLE} = \frac{(\sum_{k=1}^{p} \sum_{i \in B^{(k)}} M_i^{(k)})}{pM}$. Since $|B^{(k)}| = 2^{k-1}$, the overall running time is $O(\sum_{k=1}^{p} 2^{k-1}) = O(2^p)$. ∎

In summary, it takes $O(2^p)$ time to compute

$$\hat{b}_{MLE} = (\sum_{k=1}^{p} \sum_{i \in B^{(k)}} M_i^{(k)})/pM,$$

where M is the total item count, $M_i^{(k)}$ is the ith k-aggregate in $U_i^{(k)}$ at level k, and $B^{(k)} = \{i \mid U_i^{(k)}$ is assigned b fraction of the parent $(p-1)$-aggregate and $i \in \{0, \ldots, 2^k - 1\}\}$.

Example. For data in Figure 3.6, $M = 5$, when $p = 2$, $B^{(1)} = \{0\}$, $B^{(2)} = \{1, 2\}$, $\hat{b}_{MLE} = \frac{3+(2+2)}{2*5} = 0.7$.

Since we cannot maintain all levels p on the data stream since 2^p will exceed available memory for large p, we use only the top few levels and thus the resulting estimation error can be large.

Method 2: Pyramid Approximation.

This heuristic for estimating the parameter b is inspired by the so-called "method of moments" from statistics, and immediately follows from the definition of the b-model. It aggregates the data at multiple prefix levels and then independently finds the best fit to the fractions, at each level, based on the corresponding prefix aggregates; see Figure 3.6 for an illustration.

- **First level estimates**: M is the total packet count. For 1-bit prefix, it is easy to count packets $M_0^{(1)}$ and $M_1^{(1)}$ falling into $U_0^{(1)}$ and $U_1^{(1)}$, respectively. Without loss of generality, we assume $M_0^{(1)} > M_1^{(1)}$. Since $b \in [0.5, 1]$, this would yield two equations $M_0^{(1)}/M = b$ and $M_1^{(1)}/M = 1 - b$. Accordingly, two estimates on b-values are returned from the first-level construction.

- p**th level estimates**: Recursively, at level p, there are totally 2^p equations, one for each p-aggregate, in the form of $b^j(1-b)^{p-j} = f_i^{(p)} = M_i^{(p)}/M$, where j is the number of times an input follows the heavier edge weighted b along the path from root to $U_i^{(p)}$. Estimation at large prefix lengths p requires solving high-degree polynomial equations. We used a combination of Newton-Raphson and bisection method to find the roots of such polynomial equations when they exist.

- $\hat{b}$ **estimate**: These estimates are combined into a single estimate $\hat{b}$ by taking the average (or median) over the individual estimates in the pyramid; we use the notation $\hat{b}_{pyr}$ to denote the value obtained from this combination.

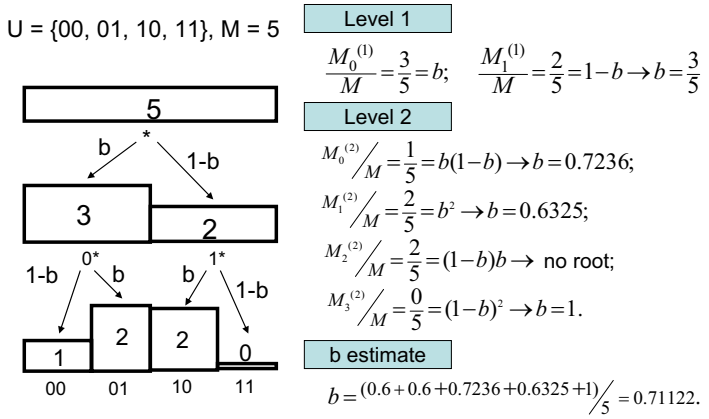


Figure 3.6: An example of pyramid approximation.

This method keeps two counters at the first level, four at the second, etc.; at any prefix level p, there are 2^p counters. When it is time to estimate $\hat{b}$, we solve an exponential (in p) number of equations and aggregates. Hence, it is infeasible

both to store this many counters and to solve as many polynomial equations for *all* levels of the pyramid. In the streaming setting, we will therefore use this method only for the top few levels (say $p = O(\log\log|U|)$). This does not provide any guaranteed error for $\hat{b}$ estimation.

3.5.2 Accuracy of Proposed Method

Parameter Estimation. For control purposes, we generated synthetic data using the stack-based algorithm in [110]. Given a bias b, data volume N, and number of levels k, this random generator returns $(val, freq)$ pairs. In our case, we set N to 10^8 and k to 32 (to simulate the IPv4 domain); b ranged between 0.55 and 0.95. We evaluated the offline method for deriving an estimate of b on the data sets to isolate out additional error resulting from sketches that are needed for online estimation, so that the effectiveness of the offline algorithm in deriving a correct estimate of b, based on correlation dimension, is evaluated. We see in Table 3.1 that $\hat{b}_2$ is almost identical to b.

b	$\hat{b}_2$	b	$\hat{b}_2$	b	$\hat{b}_2$
0.55	0.62	0.70	0.7009	0.85	0.850000
0.60	0.63	0.75	0.75007	0.90	0.900000
0.65	0.66	0.80	0.800002	0.95	0.950000

Table 3.1: Estimates of b on synthetically generated data.

We observed that the difference between b and $\hat{b}_2$ becomes smaller as b increases. Recall that the distribution becomes more biased with increasing b. For less biased data, if the data size is less than $|U|$, truncation errors of bucket counts affect the accuracy of $\hat{b}_2$.

We repeated this experiment, using the same methods, on IP addresses from real TCP packets in the AT&T network collected from a full hour and repeated the measurements over the course of multiple consecutive hours to examine the

stability of the measurements. We also tried experiments on data collected in different hours and days, and observed very similar results. We demonstrate results from one data set in depth. For these experiments, we report the results from our experiments at consecutive landmark windows of one-minute time intervals from destIP on TCP data, at a speed of roughly 60,000 packets/minute. Unlike with

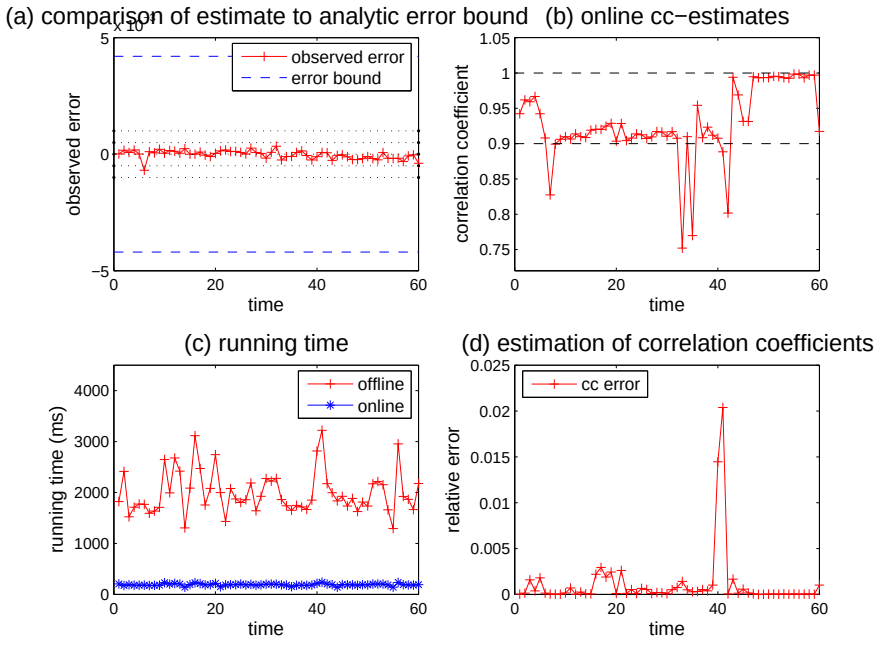


Figure 3.7: Algorithm evaluation: accuracy and running time of online and offline fractal model fitting, at per-minute time intervals.

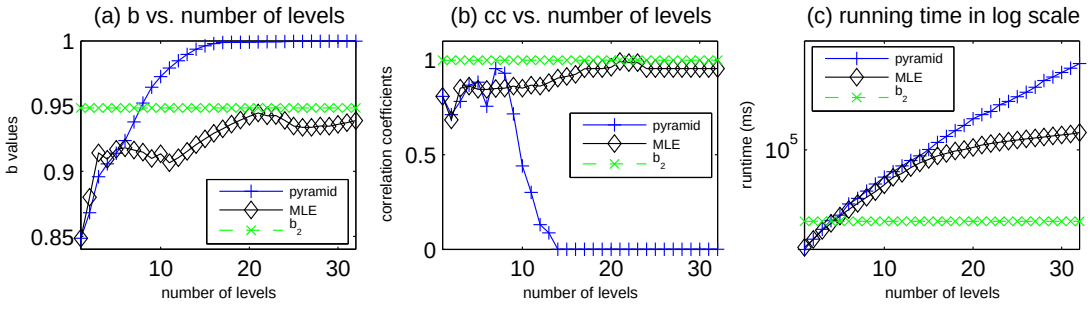


Figure 3.8: Algorithm comparison: accuracy and running time of various algorithms, as a function of the number of levels.

the synthetic data, here we do not know the distribution of the data *a priori* and, in particular, there is no known 'ideal' value for b. Therefore, we validate the

goodness-of-fit using the correlation coefficient, denoted *cc*, obtained from quantiles of the data compared to those from data that would be generated by the model (known as the q-q test and described in Section 3.3). In all experiments, we first consider the approximation error resulting from the underlying CM (the sketch width was 512 units) technique by comparing the offline $\hat{D}_2$ estimates with its online counterparts. As we showed in Section 3.3.2, the error of the approach based on $\hat{D}_2$ is bounded (with high probability). Indeed, this bound tends to hold in practice; see Figure 3.7(a). A similar trend occurs with the differences between online and offline estimates for $\hat{b}_2$, since they can be derived from $\hat{D}_2$, but we are presenting the error with respective to D_2 here because the error bounds are independent of D_2 and thus more intuitive to present graphically. We also plot the q-q test correlation coefficients from b-value estimates based on the online approaches as shown in Figure 3.7(b). Here the correlation coefficient for $\hat{b}_2$ is close to 1 at almost all time intervals, indicating a good model fitting. Moreover, running time of the online algorithm, measured by Unix `time` command, shown in Figure 3.7(c), is at least an order of magnitude faster than its offline counterpart.

Online Goodness-of-Fit Estimates. The correlation coefficients used to evaluate the goodness-of-fit of $\hat{b}$ in the experiments above were computed offline because the q-q tests require quantiles from the data stream as well as from data generated from the fractal model using $\hat{b}$. How accurate are the correlation coefficient estimates for our proposed method using approximate quantiles in comparison to those computed offline (using exact quantiles)? Figure 3.7(d) indicates that the difference is insignificant, in most cases within an absolute difference of 0.0025, and in the (rare) worst case within 0.017.

3.5.3 Comparison of Methods

We have three methods for estimating b on a data stream. The pyramid and MLE methods only explore the top few levels since the space used by them explodes

exponentially as the levels are increased. In contrast, our proposed method from Section 3.2 uses small space per level and hence is able to explore the entire depth of the hierarchy. Furthermore, the proposed method provides accuracy guarantees on $\hat{b}$.

We compared these three methods on IP addresses from TCP packet traffic data. We varied the number of levels (from 1 to 32) used to derive $\hat{b}$, but found that for pyramid approximation, this marginally improved the estimates at medium levels, and deteriorated the accuracy with too many levels; for MLE, $\hat{b}_{MLE}$ gradually converged to $\hat{b}_2$ with many levels considered. Figure 3.8(a)(b) summarize the results. For comparison, the estimates for $\hat{b}_2$ are plotted as a straight line. Here pyramid approximation yielded the worst estimates. As shown in Figure 3.8(b), the correlation coefficient for $\hat{b}_2$ is always higher than that based on pyramid approximation at any level. Pyramid method has many sources of error: approximation error from the numerical solution of high-degree polynomials, sometimes such polynomials do not have roots, limiting the number of levels to fit the space constraints, etc. In contrast, MLE is more robust. $\hat{b}_{MLE}$ starts behaving well after level 21, with stable goodness-of-fit measures thereafter. Figure 3.8(c) summarizes the running time to estimate $\hat{b}_{pyr}$, $\hat{b}_{MLE}$ and $\hat{b}_2$ in the log scale as a function of the number of levels. The limitations of pyramid and MLE methods are apparent; hence, we do not consider them in the remainder of our experiments.

3.5.4 Performance On a Live Data Stream

We implemented the proposed method, based on estimating $\hat{D}_2$ using CM sketches, as a User-Defined Aggregate Function (UDAF) facility in Gigascope [29], a highly optimized system for monitoring very high speed data streams [37]. The source used for these performance experiments was a live IP packet stream monitored at a network interface inside the AT&T network, and the whole experiment ran

for several weeks. On average, the streaming rate at this interface was about 100,000 packets/sec, or about 400 Mbits/sec. The monitoring architecture was a dual Intel Xeon 3.2 GHz CPU running FreeBSD. Our proposed method incurred only 2% CPU utilization.

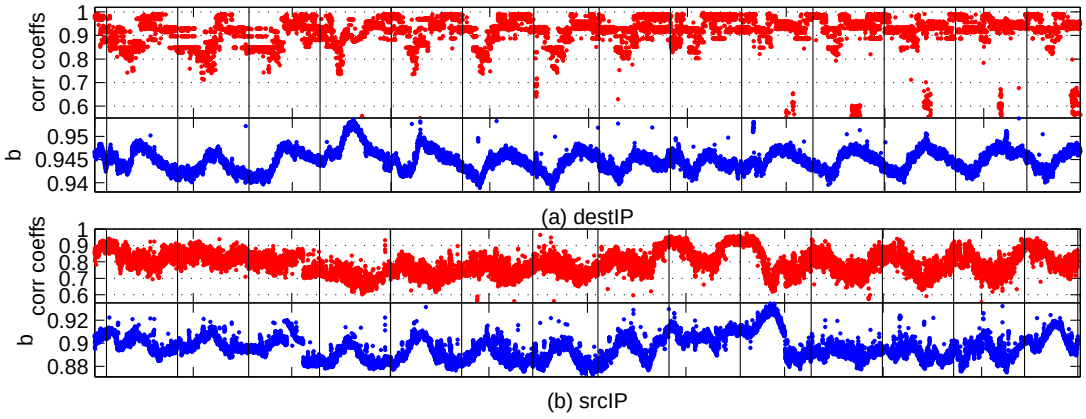


Figure 3.9: Performance and Study of Stability and Fit of Models on Gigascope: b-values, correlation coefficients. (a) TCP data on destIP; (b) TCP data on srcIP.

To evaluate the robustness of the model, we estimated the average correlation coefficient obtained from q-q plots, at landmark windows of one minute for a period of several weeks. To evaluate the stability, we estimated (a) the mean parameter value as its standard deviation; and (b) the standard deviation of correlation coefficients. Based on these measures, one may draw one of the following inferences:

- If the average correlation coefficient is low, then the model is generally not a good fit for the data and can be rejected.
- If the average correlation coefficient is high with low standard deviation, then the *model class* with generic parameter b is a good fit.
- If the average correlation coefficient is high with low standard deviation *and* the standard deviation of $\hat{b}$ is low, then the *instantiated model* with a specific parameter estimate (say, $avg(\hat{b})$) is a good fit.

We estimated $\hat{b}_2$ and its corresponding correlation coefficient $\hat{cc}$ on multiple simultaneous groupings of the data (on either srcIP or destIP addresses, and for either TCP, UDP, ICMP, or all IP packets). Figure 3.9 illustrates the time series of $\hat{b}_2$ and $\hat{cc}$ for (a) destIP and (b) srcIP on TCP data, for a period of several weeks.

Figure 3.9(a) shows a fairly robust fit for the instantiated model, with mean correlation coefficient 0.918 and standard deviation 0.0662. In addition, the estimate for b was quite stable, fluctuating within a very small range $[0.928, 0.985]$ with mean value 0.945 and standard deviation 0.002 over the interval of several weeks. Some additional observations:

- The $\hat{cc}$ values were highest when $\hat{b}_2$ was near the mean value and lowest the further $\hat{b}_2$ was from the mean in either direction.
- Such fluctuations occurred at regular cycles.
- The fluctuations of $\hat{cc}$ appear to correlate with the traffic distribution (described by $\hat{b}_2$).
- The time series was generally 'smooth', that is, the change in value of $\hat{b}$ at consecutive time steps was usually very small. However, there were occasional instances of large jumps, perhaps indicating outliers in the data.

As a 'sanity check', we tried to skew the distribution on srcIP addresses by selecting only packets originating from three different ISPs. The resulting trimodal distribution should therefore not be suitable for our model. As expected, the average $\hat{cc}$ in Figure 3.9(b) is 0.7612, and dips below 0.5 for some windows. Accordingly, $\hat{b}_2$ has a standard deviation of .011, which is an order of magnitude higher than that on destIP.

The results on UDP and ICMP packets indicate that the fractal model does

	mean(b) ± *stdev(b)*	*[min(b), max(b)]*	*mean(cc)* ± *stdev(cc)*
destIP/TCP	0.94 ± 0.002	[0.93,0.985]	0.92 ± 0.06
destIP/UDP	0.97 ± 0.007	[0.946,0.99]	0.51 ± 0.14
destIP/ICMP	0.95 ± 0.012	[0.92,0.994]	0.47 ± 0.12
destIP/all	0.94 ± 0.003	[0.935,0.98]	0.90 ± 0.06
srcIP/TCP	0.90 ± 0.01	[0.87,0.991]	0.76 ± 0.07
srcIP/UDP	0.93 ± 0.01	[0.89,0.994]	0.52 ± 0.09
srcIP/ICMP	0.96 ± 0.01	[0.91,0.995]	0.46 ± 0.08
srcIP/all	0.89 ± 0.01	[0.87,0.99]	0.75 ± 0.06

Table 3.2: Some statistics from Gigascope experiments.

not fit well, having low correlation coefficients; Table 3.2 summarizes these statistics. On the other hand, the model was fairly robust on all IP packets because TCP accounts for a very large portion of the traffic.

Our experiment demonstrates the need for online model fitting over a long time interval. Observe in Figure 3.9(a) that, although the average correlation coefficient was quite high (.918) with small standard deviation (0.0662), there were rare occasions when it dipped below .6. Had model fitting been based on a snapshot with such low q-q correlation, the model might have been rejected. Likewise, there are instances in Figure 3.9(b) where the correlation coefficient exceeds .95, though it often tends to be much lower (on average .7612 with a small standard deviation of .0698). Hence, different snapshots can lead to vastly different conclusions, so it is clearly more robust to consider a series of snapshots before drawing a conclusion.

Such insights into how the fractal model fits IP traffic data streams would not have been possible without our online algorithms since running offline methods on several weeks of trace is infeasible; even capturing and storing such a large trace is impractical.

3.5.5 Intrusion Detection Application

We obtained network traffic data tagged with known intrusion detection attacks [86]. There are several weeks' worth of IP packet header data labeled with the date, starting time, the attack name and the target destination IP addresses (typically over multiple groups of IP prefix ranges) of each attack. We chose data from a single day on which the traffic rate was roughly 10K packets per minute, estimated $\hat{b}_2$ at respective landmark windows of 3 minutes and 6 minutes (using the `destIP` field), and plotted the time series of $\hat{b}_2$ under the two window sizes; see Figure 3.10. Within each time series, we also indicate the start time of each attack with a spike. There were two types of attacks that occurred during the day we examined: 'satan' (first spike): a network probing attack which looks for well-known weaknesses; and 'neptune' (second spike): SYN flood denial of service on one or more ports.

The target destination IP address prefixes incurred an increase in traffic volume during both attacks, and the distribution of overall packet traffic shifted towards these IP addresses during the attacks. As a result, an increase in $\hat{b}_2$ was observed.

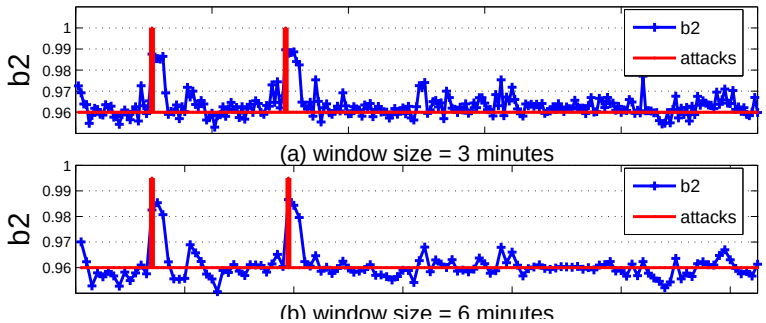


Figure 3.10: Model Fitting on IP Traffic Data with Network Intrusion Attacks

In Figure 3.10, it can be seen that: (1) the high jumps in the time series of $\hat{b}_2$ occur immediately after spikes tagged by the data source, while $\hat{b}_2$ estimated on the other time windows stays fairly constant; (2) when attacks happen, they

usually last for awhile, resulting in $\hat{b}_2$ staying high, in contrast to the small fluctuations during other intervals; and (3) at the larger window size (Figure 3.10(b)), the time series of $\hat{b}_2$ flattens out while the network is not being attacked whereas the b-values estimated during attacks remain high.

Given the fact that, under normal traffic conditions, $mean(\hat{b}_2) = 0.962, stdev(\hat{b}_2) = 0.007$, an increase in $\hat{b}_2$ to above 0.985 is a potential indicator of anomalies in this time series. This suggests that fitting the b-model parameter online has potential "discriminatory power" for intrusion detection monitoring. Still, flexible streaming algorithms like ours are needed to get a good understanding of the variability of $\hat{b}_2$ and $\hat{cc}$ over time to infer critical things about IP network traffic distribution.

3.6 Pareto Model Fitting Experiments

3.6.1 Alternative Streaming Method

We again consider the maximum likelihood estimator of tail index z. Suppose we have N observed item values (e.g. flow sizes), $x_1, \ldots, x_N$. For the Pareto model, $\Pr(X > x) = (c/x)^z$ where c is the scaling constant. So $\Pr(X = x) = c^z z x^{-(z+1)}$. Given the observed sequence, its log-likelihood function is

$$l(x_1, \ldots x_N | z) = Nz \log c + N \log z - (z+1) \textstyle\sum_{i=1}^{N} \log x_i.$$

Setting its derivative with respect to z to zero, we get

$$\hat{z}_{MLE} = \frac{N(1 + \log c)}{\sum_{i=1}^{N} \log x_i}.$$ ∎

The denominator can be computed exactly on the fly. But the difficulty is computing c. In theory, c satisfies $\sum_{x \in U} \Pr(X = x) = \sum_{x \in U} c^z z x^{-(z+1)} = 1$ since the distribution is a probability function. Hence, $c = [z \sum_{x \in U} x^{-(z+1)}]^{-1/z}$; and $\hat{z}_{MLE}$ is derived by solving the equation $z = \frac{N[1-(\log(z \sum_{x \in U} x^{-(z+1)}))/z]}{\sum_{i=1}^{N} \log x_i}$ for z. Although it is theoretically feasible to exploit numerical techniques to find its root, it is practically infeasible in the streaming case since the number of log-terms in the

numerator is linearly proportional to the domain size which is prohibitive. So, as an alternative, we adapt the commonly-used heuristic of setting $c = x_{\min}$, the minimum x-value [5], so that $\Pr(X > x_{\min}) = 1$, which satisfies the definition of a probability function, and present experimental study exploring this choice of c.

3.6.2 Accuracy of Proposed Method

In all experiments, the range of the distribution that is of interest is, by default, $a = 1\% \leq F(x) \leq 15\% = b$.

Synthetic data. For control purposes, We used a standard simulation technique, the so-called inverse transformation method, to generate values from a Pareto distribution with specified tail index z. Each experiment consisted of drawing $N = 10^8$ items from a domain of size $|U| = 10^7$. We ran our proposed method to recover z from the entire tail $[a, b]$ of the generated data sets. Table 3.3 shows that z-estimates are more accurate for z in $(0.2, 1.8)$ than at the extremes. The difference between z and $\hat{z}$ becomes smaller as $|U|$ or N increases.

z	$\hat{z}$	z	$\hat{z}$	z	$\hat{z}$
0.2	0.477	0.8	0.8002	1.4	1.366
0.4	0.428	1.0	0.9991	1.6	1.496
0.6	0.6009	1.2	1.1996	1.8	1.638

Table 3.3: Estimates of z on synthetically generated data.

Flow Data. We compared the proposed streaming method with the existing offline one on flow sizes (in terms of number of bytes per flow) using NetFlow data collected at a router over multiple consecutive hours. We selected out all sessions for which the number of packets per flow is greater than 3 to exclude TCP SYN attacks. Similar results were observed from data collected in different hours, so we reported our experimental results from one-hour data set in depth, at consecutive landmark windows of two-minute time intervals, consisting of 100K flows on average.

In all experiments, we evaluated accuracy and efficiency of our proposed online algorithm ($\hat{z}$) by comparing against two versions of offline algorithms: offline using the entire tail ($\hat{z}_{et}$) and offline using quantiles ($\hat{z}_q$). We first considered the approximation error from the underlying CM sketch (set $\epsilon = 0.1, \phi = 0.01$ in Theorem 4) by comparing the offline $\hat{z}_{et}, \hat{z}_q$ with its online $\hat{z}$. Figure 3.11(a) shows that z error(q) $= |\hat{z}_q - \hat{z}|$ is bounded by analytic error bound of $2 * 0.1/(1-0.1) = 0.22$ (with high probability), as Theorem 4 predicts. Likewise, z error(et) $= |\hat{z}_{et} - \hat{z}|$, almost identical to z error(q), is also bounded by theoretical error bound. This indicates the offline heuristic to estimate z using quantiles is sufficiently accurate to approximate $\hat{z}_{et}$. Moreover, it is clear in Figure 3.11(b) that the running time of the online algorithm, measured with the Unix `time` command, is at least an order of magnitude faster than the offline counterpart.

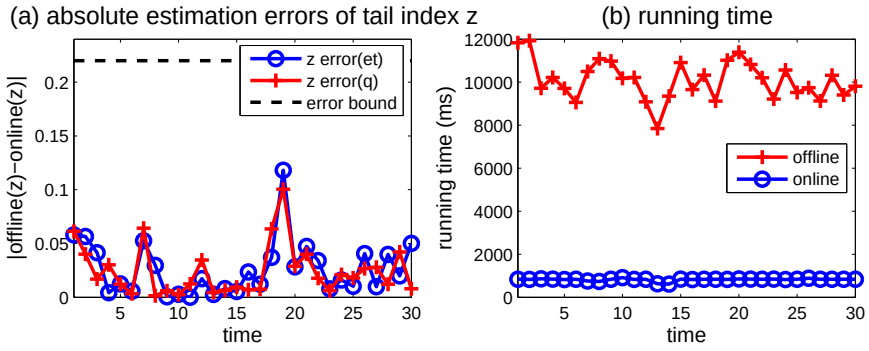


Figure 3.11: Performance of parameter estimation.

With flow data from a real network, there is no known 'ideal' value for z. Therefore, we used online validation, in particular, q-q plots, to evaluate the goodness-of-fit. We employed sequential q-q plots, each of which was generated at the same landmark windows as those used to estimate $\hat{z}$. We show one representative q-q plot (see Figure 3.12(a)) at a randomly chosen time step; the plots at other time steps look very similar. The plot clearly indicates a strong linear correlation, and thus evidence for a good fit.

3.6.3 Comparison of Methods

We repeated our experiments on flow data by comparing our proposed online method ($\hat{z}$) with MLE. As we mentioned earlier, one problem with MLE is finding the best scaling constant c, from which $\hat{z}_{MLE}$ is derived. We demonstrate the ad-hocness of picking a c for the MLE estimator by computing $\hat{z}_{MLE}$ based on three different c-values: minimum observed x-value, its minimum domain value, the average of the above two minimums, all of which satisfy the definition of a probability function (i.e. $\Pr(X > c) = 1$). We denote the respective $\hat{z}_{MLE}$ as $\hat{z}_{mo}, \hat{z}_{md}, \hat{z}_{ma}$. The average z-values, computed via different methods over all time steps, are listed below:

$\hat{z}$	$\hat{z}_{mo}$	$\hat{z}_{md}$	$\hat{z}_{ma}$
1.37	0.78	0.19	0.68

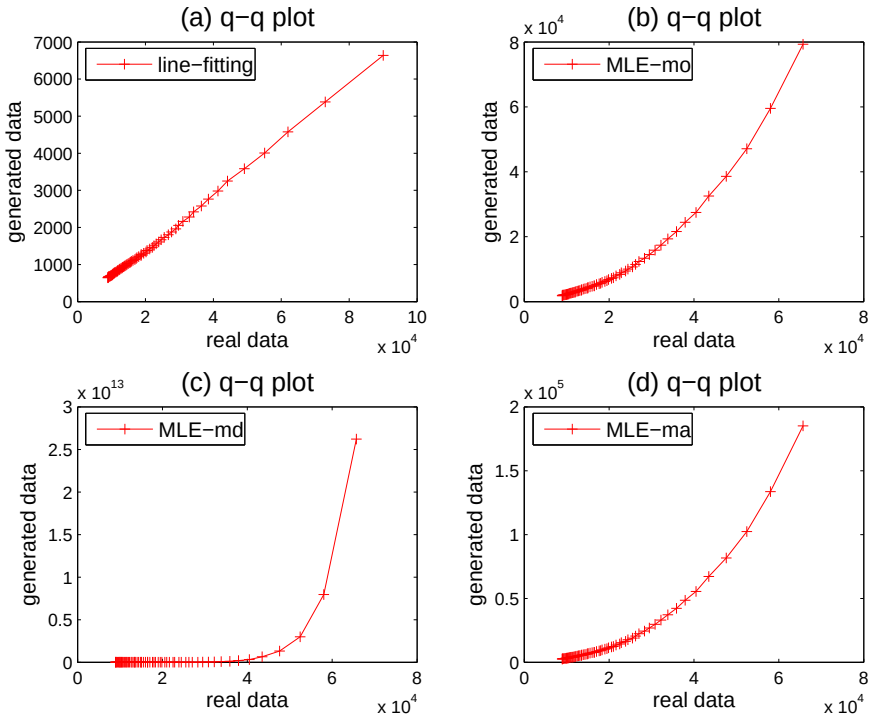


Figure 3.12: Algorithm comparison.

Note the wide disparity in z estimates between the proposed method and MLE-based approaches. We now examine their goodness-of-fits. The q-q plot for the

proposed method (Figure 3.12(a)) is clearly more linear compared to the other methods (Figures 3.12(b)-(d)) .

In addition, the above list of z estimates also tells us that $\hat{z}_{MLE}$ in general under-estimates z because of the top concavity.

3.7 Extensions

Other Streaming Windows. On the stream, our methods need to simply maintain aggregates such as L_2 norms, range sums and quantiles. We used the CM-Sketch for this purpose with the landmark window model. Our methods directly generalize to other streaming models. For example, in presence of inserts and deletes, the CM-Sketch methods dynamically maintain all these aggregates with equal accuracy as when only inserts are allowed. Similarly, with a *windowed* stream model, these aggregates can be maintained accurately using [40, 8]. Also, in the model where the past is continuously weighted down as the stream comes in, there are similar results for maintaining our desired aggregates [27]. Therefore, in all these variations of streaming windows, our methods work and give similar guarantees as the ones we show here with landmark windows. Our contribution is not in designing the synopses structures for different window models, but in showing how to use them for parameter estimation and validation, and proving accuracy guarantees in the process.

Fitting Other Models.

Hausdorff Fractal Dimension D_0. Recall notations used to define fractal model in Section 3.3.1. Set $V = \{0 \ldots, \log |U|\}$. *Hausdorff* fractal dimension D_0 is defined when

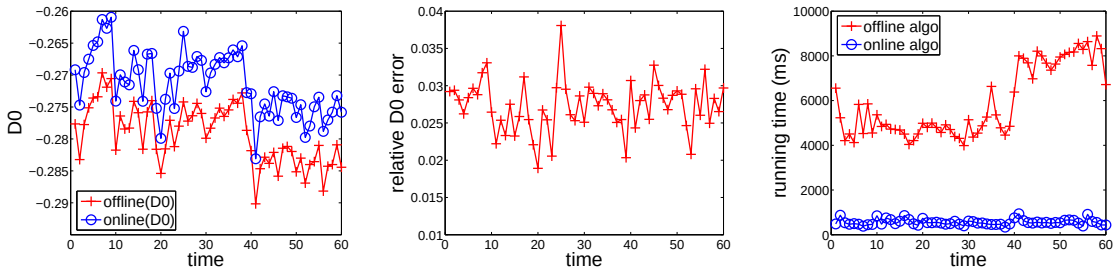
$$N(p) = \sum_{i=0}^{2^p-1} 1_{(\sum_{j=i2^{\log|U|-p}}^{(i+1)2^{\log|U|-p}-1} S[j]>0)}, \forall p \in V,$$

and $1_{()}$ is an indicator function. Instead of using the CM sketch to estimate $\log N(p)$ to derive D_2, we use hierarchical *Flajolet-Martin*, or FM, sketch [51] to

provide a $(1+\epsilon)$-approximation $\hat{N}(p)$ of $N(p)$ for each p. And D_0 and $\hat{D}_0$ are computed by the slope of the regression lines fitting $\{(p, \log N(p))|p \in V\}$ and $\{(p, \log \hat{N}(p))|p \in V\}$ respectively.

Theorem 6 *Our algorithm uses space* $O(\frac{\log|U|}{\epsilon^2} \log \frac{\log|U|}{\delta})$*, and with probability at least* $1-\delta$*, outputs estimate* $\hat{D}_0$ *for* D_0*, such that* $|\hat{D}_0 - D_0| \leq \frac{3\epsilon}{(1+\log|U|)(1-\epsilon)}$.

Proof. $\forall p \in V$, FM sketch [51] with $(\epsilon, \delta/\log|U|)$-guarantees ensures that with probability $1-\delta/\log|U|$, $(1-\epsilon)N(p) \leq \hat{N}(p) \leq (1+\epsilon)N(p)$. The rest of the proof follows the proof for Theorem 1. ∎



(a) $\hat{D}_0$ (b) relative $\hat{D}_0$ error $= |\frac{\hat{D}_0 - D_0}{D_0}|$ (c) running time

Figure 3.13: Algorithm comparison: accuracy and running time of offline and online D_0, at per-minute time intervals.

We repeat our experiments at consecutive landmark windows of one-minute time intervals from destIP on TCP data, at a speed of 60K packets per minute, and report results in Figure 3.13. We first consider the approximation error resulting from the underlying FM sketch technique by comparing the offline fractal dimension estimates $\hat{D}_0$ with its online counterpart (see Figure 3.13(a)). Figure 3.13(b) demonstrates that the relative error of $\hat{D}_0$ is smaller than 4% at all one-minute time intervals. Moreover, running time of the online algorithm, measured by Unix "time" command, shown in Figure 3.13(c), is at least an order of magnitude faster than its offline counterpart.

Information Fractal Dimension D_1. Set $V = \{0 \ldots \log|U|\}$. *Information* fractal dimension $D_1 \equiv \lim_{p\to\infty} F(p)/p$, where $\forall p \in V, F(p) = \sum_{i=0}^{2^p-1} (\frac{M_i^{(p)}}{M} \log \frac{M_i^{(p)}}{M})$.

Using appropriate sketches to estimate the entropy [21, 66, 20] hierarchically, we can obtain strong accuracy estimates for calculating the information fractal dimension as well.

Our overall approach here of (a) rigorous analysis of fitting regression line to various plots that are populated with estimations from synopses maintained over the data streams and (b) online validation using q-q plots that work without materializing the generated data, is powerful and will find many other applications.

3.8 Related Work

Parametric model fitting has a long history in the statistics literature, where the goal is to find parameter value(s) which best match the observed data. Many different methods have been proposed, from the "method of moments" to maximum likelihood estimators. There are trade-offs for different distributions to which they are applied. Our challenge here is to adapt the commonly used model fitting methods for skewed data to the streaming scenario.

There has been plethora of methods for managing data streams in general [96]. They have primarily focused on clustering (k-means or k-medians), summarizing (quantiles or histograms or wavelets) or mining (heavy hitters, change detection). In contrast to the bulk of this literature, our focus is on modeling of data streams using parametric models.

A number of different observations have been made about IP network traffic. For example, IP traffic tends to have bursty behavior [81]; it tends to have heavy-tailed distributions [112] and exhibit self-similarity in multiple scales [61]. There is a hierarchical component to their distribution over the IP address space that resonates with the hierarchical assignment of IP addresses to domains [26]. On the other hand, non-hierarchical skew of many real data sources is observed and well modeled by heavy-tailed—in particular, Pareto—distributions [112]. One

needs suitable models (hierarchical and non-hierarchical), preferably with a small number of parameters, to describe the inherent characteristics of IP traffic [114].

Regarding hierarchical model fitting, the most relevant work is [81], where they identified IP traffic streams as having the fractal behavior (using offline methods) and demonstrated that the fractal dimension of traffic over the IP address hierarchy can serve as a "fingerprint" for traffic at different sites and times. From an algorithmic perspective, a highly relevant work is [110], where the authors studied the b-model and showed how to estimate b. Their algorithm was focused on temporal modeling of IO systems and does not work on IP traffic streams because it requires that values arrive in sorted order. Another related algorithmic result is by [115], where they used the tug-of-war sketch to estimate the fractal dimension. However, their approach does not provide error guarantees, which is our crucial focus, and requires more space and update time than ours. Likewise, a lot of work exists for (offline) fitting of Pareto models [12, 102, 114, 112]. However, we are not aware of any prior work that exists on fitting Pareto distributions in data streams.

We build on the existing literature in two key ways. First, all our methods are streaming, that is, work online within small space and provide the first provable online guarantees for the accuracy of parameter estimate(s); in contrast, existing analyses are offline. Second, we introduce online validation of these parameter estimates, which was not present in previous work and is crucially needed in the data stream context where distributions and models change over time. Much of our insights into real live IP streams in Sections 3.5.4 and 3.5.5 would have been infeasible to get with the offline methods.

3.9 Chapter Summary

There has been a lot of work on techniques for analyzing streaming data, but very little has focused on modeling the data. We presented a highly efficient approach for (a) estimating paramaters of (non-) hierarchical models on streams such as IP traffic with guaranteed accuracy, and (b) online model validation, all within small space and fast update time per new data stream item.

Statistical modeling is a difficult task and it needs careful, systematic and rigorous real life experimental study to be robust. We performed detailed study of our methods on an operational DSMS (AT&T's Gigascope). Fortunately, our approach only relied on maintaining CM-Sketches on the data stream and this was already part of Gigascope for other operational reasons; hence, our approach does not add any additional burden on the system. Also, we benefited from a number of optimizations in Gigascope for this primitive. As a result, our detailed study of modeling on several weeks of IP traffic streams reveals many insights on its applicability and its use. In particular, our online validation approach proved crucial and we believe it will be of fundamental interest in DSMSs for modeling streams.

Algorithm 1 GENERATEDQUANTILES(cmh,b,ϕ) Stack-based method to generate ϕ-quantiles Y directly from b-model

```
1: percent ← 0 /*percentage of data generated*/
2: addr ← 0 /*item ID*/
3: i ← 1 /*next quantile*/
4: push a pair (1,0,0)
5: while stack not empty do
6:    pop a tuple (frac, level, index) from the stack
7:    if level == log |U| then
8:       percent+ = frac
9:       while (percent >= iφ) and (i < 1/φ) do
10:         Y[i] = addr; i++
11:      end while
12:      addr++
13:   else if percent + frac <= iφ then
14:      /* prune the subtree: no quantiles*/
15:      percent+ = frac; addr+ = 2^(log |U| − level)
16:   else
17:      /*assign weights according to range queries*/
18:      w_l = CMH_Count(cmh, level + 1, 2 ∗ index)
19:      w_r = CMH_Count(cmh, level + 1, 2 ∗ index + 1)
20:      if (w_l < w_r) then
21:         /*left (1-b), right b*/
22:         if percent + frac ∗ (1 − b) < iφ then
23:            /*prune left branch: no quantiles*/
24:            percent+ = frac ∗ (1 − b)
25:            addr+ = 2^(log |U| − level − 1)
26:            /*push the right branch*/
27:            push (frac ∗ b,level + 1,2 ∗ index + 1)
28:         else
29:            push (frac ∗ b,level + 1,2 ∗ index + 1)
30:            push (frac ∗ (1 − b),level + 1,2 ∗ index)
31:         end if
32:      else
33:         /*left b, right (1-b)*/ symmetric with L21-31, exchange b and 1 − b
34:      end if
35:   end if
36: end while
37: return Y /*φ-quantiles*/
```

Chapter 4
Modeling Communication Graphs via Signatures

4.1 Introduction

In the everyday world, instances of interaction or communication between individuals are everywhere. For example, individuals speak to each other via telephone; IP traffic is passed between hosts; authors write documents together and so on. There are other examples in which individuals interact with other entities, such as when users pose search queries; post and comment on messages on bulletin boards or blog sites; or when stock traders transact stocks, bonds and other goods. In an indirect sense, users "communicate" with each other via the common objects. In all cases the communication between individuals can be repeated many times (such as in the case of telephone calls) and weighted (say, the quantity of stock bought, or the duration of a call).

Given this abundance of communication between individuals, many applications rely on analyzing the patterns behind the communications, for example:

- *Anti-Aliasing:* is an individual behind multiple presences in the communication network? This happens e.g. when an individual has multiple connection points (home, office, hotspot) to the Internet.

- *Security:* has some individual's 'identity' been taken over by someone else? This happens when a person is given access to another's laptop, or when a cellphone is stolen and used by someone else. Is a new user who arrives at a particular time really the reappearance of an individual who has been

observed earlier? This happens in telephone networks when a consumer defaults on an account and opens a fresh account to further use services without paying (a "repetitive debtor"); or in bulletin boards when a banned user re-registers with a new ID.

- *Privacy Preservation:* can we identify nodes from an anonymized graph given outside information about known communication patterns per individual? This happens in author identification of double-blind submissions.

Each of the questions above and others of this nature that rely on communication patterns can naturally be solved by designing suitable *signatures* for the individuals. Informally, signatures capture the distinctive or discriminatory communication behavior of an individual (telephone user, IP address, trader or user of a search service, etc). While the concept of signatures is self-evident, formalizing and applying signatures to a specific task is really an art. Typically, in any particular task, "signatures" are defined based on intuition and experimentally validated against a labeled set. This approach has been instantiated successfully for certain specific communication settings and applications. It was done for telephone networks in [36, 68] where the authors defined a *community of interest* to be the top-k numbers called by a given telephone number. With appropriate age weighting and a suitable k, this was argued to be highly discriminatory for detecting repetitive debtors. A second example is when a signature formed from bibliographic citations is used to identify authors of double-blind submissions [69]. There are many other examples in areas including Security [83, 48, 35, 36], Networking [99, 117, 85, 98], Social Network Analysis [59, 15, 107], Epidemiology and others.

In this chapter, we adopt the signature-based approach to analyzing the patterns of communication exhibited by individuals. However, we focus on the process of how signatures are developed and applied. In prior works, typically, one

considers a particular application, proposes an intuitive signature and evaluates it empirically on datasets. We focus on principles behind the usage of signatures. In particular, we propose the framework in which we first agree on a set of properties of signatures that are natural, and when faced with an application, determine what properties of signatures are needed, and then, seek out examples of signatures already known or design new ones which will have those properties. Hence, the process will focus on abstract properties that are needed, and "shopping" for signatures with those properties.

We develop a framework for the formal use of signatures for tasks that involve analyzing communication patterns, for very general notions of communication. We model the communications between entities using a suitably weighted graph, and define a signature of a node abstractly in terms of the graph. Our contributions are:

- We identify basic properties of signatures such as persistence, uniqueness and robustness, and for several tasks that involve analyzing communication patterns, study which of these properties are needed. For example, a task such as finding multiple presences of the same individuals in a time window does not need signatures to be persistent; likewise, analyzing the changing behavior of a single individual over time may not need signatures to be discriminating.

- We consider specific signatures — some previously known, others new — for communication graphs and study what basic properties they have, using extensive experiments with real datasets. This helps identify which signatures are suitable for each of the tasks.

- We complement our conceptual results with a detailed experimental study on two concrete applications of enterprise network traffic. We adopt the framework for respective tasks. Our results show that signatures based on

a combination of a few application-desired properties is quite effective for us.

It remains the case that finding suitable signatures for any task is more of an art than a science, with effectiveness determined experimentally. In this sense, our proposal of specific signatures for communication graphs and their application to the specific tasks is such a study. But beyond this, our framework is general and can be applied broadly.

In what follows, we first introduce our framework for analyzing signatures in Section 4.2, the properties we desire, and an analysis of their values for a variety of applications. In Section 4.3, we describe various signature schemes based on expected features of communication graphs, and their characteristics. We evaluate signatures empirically, first studying the general characteristics in Section 4.4, and then for particular applications in Section 4.5. We lastly discuss scalability issues in Section 4.6, then survey related work and give concluding remarks.

4.2 Framework

Here we describe our framework for designing and evaluating topological signatures for communication graphs. We define the domain of our signatures, and three general properties for evaluating them. Finally, we discuss how these properties relate to specific applications of signatures.

4.2.1 Individuals and Labels

A communication graph is defined by the observed patterns of communications between nodes representing individual users. However, we observe only the *labels* of these nodes rather than the actual identities of the *individuals* who are communicating. For example, we may see traffic on a network between pairs of IP

addresses, or calls between pairs of telephone numbers. These may be unique to individuals, but not necessarily: an IP address may be dynamically reassigned to another user, a cell phone may be loaned to a friend, etc.[1] What we can do is to analyze the observed communication between nodes in the graph, and infer the behavior of individuals. We need the assumption that the hidden mapping of individuals to node labels in the graph is for the most part consistent over time: if the mapping of every label is randomly reassigned at every time step, then the task of building good signatures becomes appreciably harder, especially if only basic information about the communications is available. Indeed, many of the applications we discuss here concern finding examples where the mapping from users to labels is slightly perturbed. In subsequent sections, we concentrate on building signatures based on the observable labels, while understanding that our purpose is to use the signatures to identify the behavior of individuals.[2]

4.2.2 Signature Space

Let $G_t = \langle V, E_t \rangle$ be a communication graph that has been aggregated over some time interval at t.[3] The graph may be revealed as a sequence of directed edges (v, u), and then aggregated, or may arrive as a set of aggregated edges. An edge $(v, u) \in E_t$ represents communication exchanges from node v to u in G_t, and the weight of edge (v, u), denoted $C[v, u]$, reflects the *volume* (e.g., frequency) of this communication. For each node $v \in G_t$, we denote by $I(v)$ and $O(v)$ the set of v's in-neighbors and out-neighbors during the time interval, respectively. That is, $I(v) = \{u|(u, v) \in E_t\}$ and $O(v) = \{u|(v, u) \in E_t\}$. In many common cases the nodes are partitioned into two distinct classes, such as clients and servers,

[1] However, we consider a group of people sharing a node, e.g., a family with a shared Internet connection, or even a computer program with a particular communication pattern, to represent an "individual" in our setting if the group membership is consistent over time.

[2] We use the terms "individuals" and "users" interchangeably; likewise "labels" and "nodes".

[3] In practice, $V = V_t$ as it may vary between windows, but only by a small amount.

and so the induced graph is *bipartite*. A bipartite communication graph $G_t = \langle V_1 + V_2, E_t \subseteq V_1 \times V_2 \rangle$, with nodes partitioned into disjoint sets V_1 and V_2, has directed edges $(v, u) \in E_t$ with $v \in V_1$ and $u \in V_2$.

To define our signatures we make use of a *relevancy function*, w, so that w_{vu} indicates the relevance of u to v. Initially, assume w is given; we later discuss choices of w.

Definition 1 (Graph Signature) *We define a communication graph signature $\sigma_t(v)$ for node $v \in V$ at time t as a subset of V with top-k associated weights*[4]*, that is,*

$$\sigma_t(v) := \{(u, w_{vu}) | u \neq v \in V, w_{vu} \geq w_v^{(|V|-k)}, w_{vu} \in \Re^+\},$$

where $k < |V|$; $w_v^{(i)}$ is the ith order statistic of $\{w_{vu} | u \in V\}$, that is, $w_v^{(1)} \leq w_v^{(2)} \leq \ldots \leq w_v^{(i)} \leq \ldots \leq w_v^{(|V|)}$.

Where the graph is bipartite, we may restrict the signature for nodes in V_1 to consist only of nodes in V_2, especially if the size of the sets is unbalanced, i.e. $|V_1| \ll |V_2|$. Otherwise, the treatment of bipartite graphs is the same as that for general graphs.

We deliberately restrict the scope of the signature space to include only graph features. Although some prior work on related questions has used features which do not fit into this setting, such as the maker of the cellphone or the age of the blog user associated with a node, and those based on interarrival distributions [73], this definition is sufficiently broad to capture a large class of possible signature schemes. In many common settings only communication "flows" are revealed, in the form of graph edges aggregated over multiple occurrences and summarized as total volumes, such as Call Detail Records in telephony [36] and NetFlow for summarizing IP traffic at a router [97]. In addition, this definition conforms

[4] *The* top *weights follow naturally since w quantifies node relevance, and thus filters out noise.*

with prior work in [36]. Thus, this restriction allows us to thoroughly explore signature schemes in a well-defined, useful space. Moreover, this definition lends to more human comprehensible signatures, and simple descriptions of causes for differences.

The above definitions leave room for many alternatives, based on the choice of w. Designing a good signature requires much insight and care. We discuss how to select an appropriate set of nodes with associated weights ("signature scheme") in Section 4.3. Next we introduce some general properties that are desirable for any signature, and discuss how they apply to a variety of problems.

4.2.3 Signature Properties

The traditional function of a signature is to authenticate an individual's identity via handwritten depictions of their name. In our context, signatures are based on profiling interactions specific to the individual. As with the handwritten case, a useful communication signature should satisfy the following properties:

Definition 2 (General Properties)

- Persistence*: an individual's signature should be fairly stable across time, that is, not differ much when comparing similarities at consecutive time intervals. Otherwise, it will not give a reliable way to identify the individual. (Slowly evolving signatures may be acceptable but abruptly changing signatures are not.)*

- Uniqueness*: one individual's signature should not match another's. That is, if two signatures match, then they should belong to the same individual.*

- Robustness*: the ability to identify an individual from a signature should not be sensitive to small perturbations. Any noise introduced in the process of deriving signatures should not interfere with its effectiveness.*

To measure these properties and so be able to compare different signature schemes, we need a way to match identities based on signatures. A natural approach involves defining distance functions $\text{Dist}(\sigma_1, \sigma_2)$ between two signatures σ_1 and σ_2. Then we can more precisely define and measure persistence in terms of the distance between a node's signature at two different time steps; uniqueness in terms of the distance between a given node's signature and that of another node in the graph; and robustness as the distance between a node's signature with and without small perturbations. That is, for a fixed v we measure the three graph properties, given some node $u \neq v$, as follows (w.l.o.g., fix $0 \leq \text{Dist}(\cdot, \cdot) \leq 1$):

- Persistence: $1 - \text{Dist}(\sigma_t(v), \sigma_{t+1}(v))$;
- Uniqueness: $\text{Dist}(\sigma_t(v), \sigma_t(u))$;
- Robustness: $1 - \text{Dist}(\sigma_t(v), \hat{\sigma}_t(v))$, where $\hat{\sigma}_t(v)$ has been slightly perturbed from $\sigma_t(v)$.

These definitions can accommodate different choices for Dist and $\hat{\sigma}_t(v)$. We can now compare different signature schemes with respect to persistence, uniqueness and robustness using distance measures. These are defined so that a larger value in each case indicates greater presence of these properties, up to 1 (perfect).

Because of the hidden mapping from individuals to labels, some trivial signature schemes do not suffice. We could assign each node v the signature $\sigma(v) = \{(v, 1)\}$: the signature is the node label. It would seem that such a scheme will be persistent (since the signature never changes) and unique (since no two nodes have the same signature). However, this is not the case since the signature relates only to the node, and not the individual: if the user changes, the signature of the node remains the same and so it fails.

Applications	Persistence	Uniqueness	Robustness
Multiusage Detection	Low	High	High
Label Masquerading	High	High	Medium
Anomaly Detection	High	Low	High

Table 4.1: Different applications and their requirements

4.2.4 Applying Signatures

We now specify some example tasks that involve analyzing communication patterns. We describe one task per category (introduced in Section 4.1) and discuss which properties of a signature (listed above) are needed to solve it. Table 4.1 summarizes these observations.

Multiusage Detection (Anti-Aliasing). Multiusage is when similar behavior is exhibited by multiple node labels simultaneously. This could be the result of malicious behavior such as in link spam where websites attempt to manipulate search engine rankings through aggressive interlinking to simulate popular content, or benign behavior such as a single individual communicating from multiple distinct node labels (e.g., "multihoming"). The key signature property needed is uniqueness, since the assumption is that if nodes have distinct users then they have dissimilar signatures. To detect multiusage, we compute $\text{Dist}(\sigma_t(v), \sigma_t(u))$ for node pairs within the tth time window, and look for high degrees of pairwise similarity.

Label Masquerading (Privacy Preservation). Label masquerading occurs when one user switches all their communication from one node to originate from another. An example of this is the repetitive debtors problem [68], where a consumer switches accounts with no intention of paying for their usage. The key signature properties required here are persistence and uniqueness. On the assumption that such masquerades are relatively rare within the whole graph, to find instances we seek node pairs where there is very little or no similarity within one time window of interest, but very similar behavior in subsequent

windows. Formally, the detection process involves computing the persistence values $1 - \text{Dist}(\sigma_t(v), \sigma_{t+1}(u))$, for each v, and uniqueness values of a fixed v $\text{Dist}(\sigma_t(v), \sigma_t(u))$, for each $u \neq v$. A masquerader who switches from v to u is likely to be detected when corresponding persistence and uniqueness values are both high.

Anomaly Detection (Security). We define an anomaly as an abrupt and discernible change in the behavior of a fixed label v observed in consecutive time windows. This change could be the result of malicious behavior such as fraud, or could be due to benign factors such as one individual going on vacation (and so changing their communication pattern). The key signature property that will be useful for detecting anomalies is persistence. Robustness is also needed, as we expect some noise and variations over time. Uniqueness is not as important here: we can tolerate some nodes have similar signatures, since we only compare signatures of the same node over time. A simple algorithm to detect anomalies from signatures is to compute value given by the above definition of persistence, $1 - \text{Dist}(\sigma_t(v), \sigma_{t+1}(v))$, for each v, and reporting those v with unusually small values. Consequently, signatures that exhibit higher persistence over a longer term will be more effective at detecting anomalies.

4.3 Example Signature Schemes

The framework in Section 4.2 leaves a lot of scope for different signature schemes that satisfy the desired properties. In this section, we study different features of communication graphs that help us build useful signatures.

- *Engagement/Communication strength*: the edge weights in communication graphs indicate the amount of interaction between each pair. So a heavier edge should make the participating pair of nodes "closer" to each other,

and hence more likely to figure in each other's signatures. Basing signatures on these larger weights should make the signatures robust to small perturbations. Further, we can assume that high interaction in one time period predicts high interaction in future time periods, and so will improve persistence.

- *"Novelty" of neighbors*: typically communication graphs exhibit a "power-law"-like distribution of node degrees, so a few nodes have very high degree, but the majority have smaller (constant) degree. A node with high in-degree in a graph may be a poor member of a signature, since it is not very discriminating. For example, a directory assistance number in the phone graph or a search engine in the web traffic graph may be used by many people, and hence be poor in distinguishing between them. So nodes with lower in-degree are more "specific", and may be preferable for uniqueness.

- *Locality*: because of the degree distribution, communication graphs are far from complete, and instead some nodes are much closer (in terms of graph hop distance) than others. For a given node, choosing nearby nodes may be more relevant than those that are far away, leading to increased distinguishability and hence uniqueness. In addition, a signature may be more human interpretable if it relates to nodes in the immediate neighborhood than seemingly arbitrary nodes scattered across the whole graph.

- *Transitivity/Path Diversity*: communication graphs, although not dense, are also far from being skeletal trees; between pairs of nodes there are typically many paths. We assume that the more connecting paths, the "closer" these two nodes are (even if they are not directly connected). That is, a signature is likely to be more persistent and robust if it relates node pairs with multiple connecting paths.

Characteristics	Properties
Engagement	persistence, robustness
Novelty	uniqueness
Locality	uniqueness
Transitivity	persistence, robustness

Table 4.2: Communication Graph Characteristics and Properties

A signature scheme which incorporates some of these features can lead to greater presence of the desired features. Table 4.2 summarizes the links between graph characteristics and our desired signature properties.

We now describe a variety of signature schemes. Most are quite simple to state, and based on extensions of prior work. We emphasize that our concern is not the novelty or otherwise of these signatures, but rather the evaluation within our framework, and the extensive experimental comparison which follows.

4.3.1 One-hop Neighbors Based Approaches

We first consider signature schemes that only pick from the immediate (one-hop) neighbors in the graph. For each neighbor j of $i \in V$, we compute a relevance measure w_{ij}, indicating the computed importance of j to i. Following Definition 1, we retain the k nodes j with the largest values of w_{ij}. For bipartite graphs, for each $i \in V_1$, we retain the k nodes j among V_2 with the largest values of w_{ij}. Ties may be broken arbitrarily, and if there are fewer than k nodes with non-zero values of w_{ij}, we retain only this subset. To ease our discussions in the rest of the chapter, we shall not explain explicitly different notations used for bipartite and non-bipartite graphs, like what we do here, but keep notations following Definition 1 when the context is clear. We consider two such schemes:

Definition 3 *The* Top Talkers (TT) *scheme sets* $w_{ij} = C[i,j] / \sum_{(i,v) \in E_t} C[i,v]$. *That is, the signature of* i *consists of the (at most)* k *nodes adjacent to* i *with the highest incoming edge weights* w_{ij} *from* i.

This might correspond to the most called telephone numbers, or the most visited web sites, for i. The definition only takes into account Communication Strength, and is implicit in the "Communities of Interest" work, which used such signatures in the course of detecting fraudulent activity [36]. A feature of that work was that it additionally created a signature from the combination of multiple time-steps by using an exponential decay function applied to older data. It is straightforward to apply these definitions over a set of modified edge weights $C'[i, j]$, which reflect an appropriate exponential decay or other combination of historical data. Hence, we treat such time decay as orthogonal to our main line of inquiry, and do not consider it explicitly any further.

Definition 4 *The* Unexpected Talkers (UT) *scheme sets* $w_{ij} = C[i, j]/|I(j)|$. *Thus the signature for* i *consists of the (at most)* k *nodes* j *with the largest incoming edge weights from* i*, scaled by the number of* j*'s incoming edges.*

By factoring in "Novelty" of neighbors, this definition downweights nodes which might be universally popular and dominate signatures, leading to false matches and hence low uniqueness. The prevalence of such nodes will depend on characteristics of the setting inducing the communication graph. For example, there are relatively few nodes of this kind in the telephone call graph: although people may regularly call directory assistance, they will typically call friends and family more often, hence such nodes are unlikely to dominate their signature. However, in the web traffic graph, one can observe sites which attract a lot of incoming traffic, from many different users, such as search, web mail, and video sites. Having such nodes in a signature is unlikely to provide a good signature. One could remove such nodes altogether. We avoid this for two reasons. Firstly, there can be many such nodes, and the list is not static, but evolves over time as new nodes attract interest. Secondly, there is still some information in the set, such as relative weight of such nodes in the communication pattern of a particular node, and we want to create some signature even if these are the only

destinations a node i communicates with. So we downweight this, to try to push up more "unexpected" destinations, and hence create a signature that is more likely to be unique. In full generality, we can consider many possible settings for this downweighting: we can use many functions of $|I(j)|$ and $C[i, j]$ (e.g., $C[i, j] \log(|V|/|I(j)|)$, by analogy with the TF-IDF measure). In our detailed experiments, we did not see much variation in results for different scaling functions, so we focus on this definition for brevity.

4.3.2 Multi-hop Neighbors Based Approach

The one-hop approach is highly appropriate for certain graph types, in particular the telephone call graph. But we can conceive of other communication graph settings where no one-hop signature can do well. Consider the (bipartite) communication graph induced by customers renting movies. If we consider two subsequent time periods, it is highly unlikely that any significant fraction of customers will rent the same title in both periods. Thus, no matter what weighting of their one-hop neighbors (i.e. their rentals that month) we apply, we will obtain signatures with poor persistence. In general, this is a challenging situation to make signatures for; however, we can at least hope for somewhat better signatures if we look beyond the immediate neighborhood.

For a multi-hop signature based approach to be successful, we need to be able to find nodes and weights outside the immediate (one-hop) neighborhood of node i that nevertheless accurately represent i. So, even if i communicates with completely different sets of nodes in each time period, our hypothesis is that there is sufficient information in the broader link structure of the graph so that we will find a set of nodes and weights for i that are similar in both time periods (persistence) while being different to those found for other nodes (uniqueness). Clearly, the validity of this will depend on the nature of the communication graph. We propose an example signature scheme, and validate it experimentally

on a variety of graphs.

Definition 5 *The* Random Walk with Resets (RWR) *signature scheme is defined as follows: starting from node i, we define $\vec{w}_i = [w_{ij}]_{|V|\times 1}$ as the steady-state probability vector, where w_{ij} is the probability that a random walk from i occupies node $j \in V$. Each step in the random walk either selects an edge to follow with probability proportional to the edge weight or, with probability c, returns to node i.*

As before, we take the k largest w_{ij}s in $\vec{w}_i$ to define the signature for i. Although this is the stationary distribution of a random walk, it is can be computed exactly. The definition of w_{ij} is equivalent to the personalized PageRank [67] with an input set of preferences equal to the single node i and can be computed as follows.

Computation of RWR. Recall that C is the adjacency matrix of the graph G_t from which we compute the transition matrix P. Here $P(i,j) = C[i,j]/\sum_{j=1}^{|V|} C[i,j]$ denotes the probability of taking edge (i,j) from node i. Let $\vec{s}_i$ be the start-node vector with 1 in position i and 0 elsewhere. Then the steady-state probability vector $\vec{r}_i$ satisfies

$$\vec{r}_i = (1-c)P\vec{r}_i + c\vec{s}_i, \tag{4.1}$$

where $\vec{r}_i$ is initialized to $\vec{s}_i$ and c is the probability of resetting. This equation can be solved by

$$\vec{r}_i = c[I - (1-c)P]^{-1}\vec{s}_i. \tag{4.2}$$

Then $\vec{r}_i$ can be found for any i if we pre-compute and store $[cI - (1-c)P]^{-1}$. However, this becomes impractical for large graphs due to the high cost of computing the inverse. So we use the iterative approach by applying (4.1) repeatedly: $\vec{r}_i^{\,l} = (1-c)P\vec{r}_i^{\,l-1} + c\vec{s}_i$. This is guaranteed to converge [17], and the process can be terminated after a fixed number of iterations, or after the probability vector does not change significantly. The running time is linearly proportional to the number of iterations and the number of edges, per starting position i.

Scheme	Characteristics	Properties
TT	locality, engagement	uniqueness, robustness
UT	novelty, locality	uniqueness
RWR	transitivity, engagement	persitence, robustness
RWR^h	locality, transitivity	persistence, uniqueness, robustness

Table 4.3: Properties Used by Signature Schemes

Computation of $\mathbf{RWR}_c^h$. As discussed earlier, we may wish to restrict the signature of a node to its local neighborhood. The RWR signature can pick arbitrary nodes from the graph, and may be drawn to "attractors" (nodes with high indegree). Instead, we propose RWR_c^h as a modification of the above procedure when the random walk is restricted to visit only nodes within at most h hops of i. As observed in [107], the structure of the graph means that such signatures may additionally be faster to compute. To compute RWR_c^h, we take the iterative algorithm defined above, and proceed for only h iterations. This ensures that signatures for node v contain only nodes at most h hops away from v. Notice that when $c = 0$ and $h = 1$, RWR^h is identical to the Top Talkers scheme. By increasing h, we tradeoff between the local (TT) scheme and the global (RWR) scheme.

Table 4.3 summarizes the schemes in terms of communication graph characteristics exploited and the resulting signature properties from Section 4.2.3 that are captured. Based on our analysis of application requirements, we reason that RWR will perform well at anomaly detection; RWR^h will succeed at label masquerading, and TT will be good for multiusage detection.

4.4 Evaluations of Signature Properties

In this section, we evaluate the quality of signature schemes on various data sets with respect to persistence, uniqueness and robustness. In particular, we focus on two real data sets: flow data from an enterprise network and database query

logs. All our experiments were performed on a dual 2.8GHz desktop machine with 2GB RAM. From each graph, we select signatures for each individual using the TT, UT and RWR schemes outlined in Section 4.3.

4.4.1 Data Sets

Enterprise network data. We collected six weeks' worth of flow records from a large enterprise network. LAN switches and a Network Interface Card were configured to monitor all traffic from more than 300 local hosts including desktop machines, laptops and some servers; these hosts are the focal point of our analysis. We captured all outgoing flows from the local hosts to external hosts in the network. No communications between local hosts are visible on the monitored links. In this study we used TCP traffic only, and removed weekend data from our data set for purpose of a more consistent per-day traffic mix. The total six week collection yielded more than 1.2 GB of network flow records, and contains about 400K distinct IPs. The flows were aggregated over regular time windows to form communication graphs. We used an interval of five days to present results; the results were similar with other window sizes. The weight of a directed edge was measured as the total number of TCP sessions during the time interval. In all experiments, we used the signature length of $k = 10$,[5] which is half of the average local host's out-degree.

User query logs. Our second data set consisted of 820K tuples summarizing a set of queries issued by users to a data warehouse. The logs recorded which tables were queried, but not the attributes accessed within each table. The data contains 851 distinct users and 979 distinct tables. Given a sequence of (userID, tableID) "edges", we split the trace into windows covering five consecutive time periods. Here, the edge weight is the number of times that the user accessed the

[5]Due to space limitations, we omit discussion about how we chose k. This issue was investigated in [68], and is beyond the scope of this chapter.

table within the time period. In all experiments, we used a signature length of $k = 3$, half the average number of tables a user accessed per period.

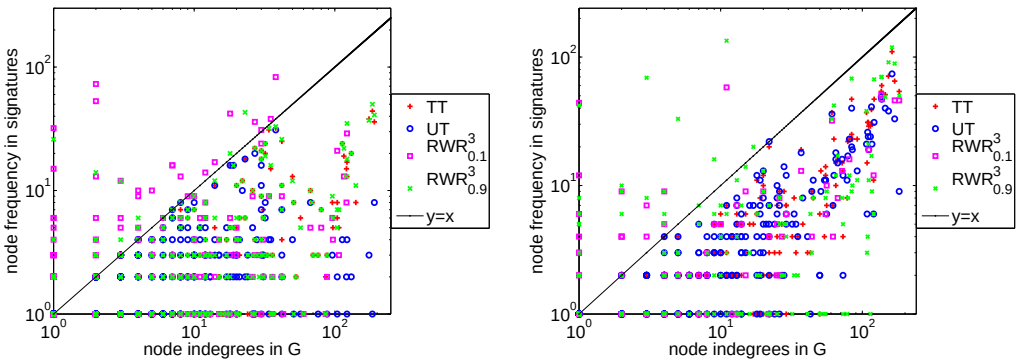
4.4.2 Distance Functions

In our evaluation, we employed a variety of distance functions to compare signatures.[6] They are generalized from known measures, and take into account both set overlap as well as weighted occurrence. Formally, given two signatures σ_1 and σ_2, where $\sigma_i = \{(u_{ij}, w_{ij}) | j = 1..k_i\}$ is of length k_i, let $S_i = \{u_{ij} | j = 1..k_i\}$ be the set of u's in σ_i. We considered four distance functions:

$$\begin{aligned}
\mathrm{Dist}_{\mathrm{Jac}}(\sigma_1, \sigma_2) &= 1 - \frac{S_1 \cap S_2}{S_1 \cup S_2}; \\
\mathrm{Dist}_{\mathrm{Dice}}(\sigma_1, \sigma_2) &= 1 - \frac{\sum_{j \in S_1 \cap S_2} (w_{1j} + w_{2j})}{\sum_{j \in S_1 \cup S_2} (w_{1j} + w_{2j})}; \\
\mathrm{Dist}_{\mathrm{SDice}}(\sigma_1, \sigma_2) &= 1 - \frac{\sum_{j \in S_1 \cap S_2} \min(w_{1j}, w_{2j})}{\sum_{j \in S_1 \cup S_2} \max(w_{1j}, w_{2j})}; \\
\mathrm{Dist}_{\mathrm{SHel}}(\sigma_1, \sigma_2) &= 1 - \frac{\sum_{j \in S_1 \cap S_2} \sqrt{w_{1j} \cdot w_{2j}}}{\sum_{j \in S_1 \cup S_2} \max(w_{1j}, w_{2j})}.
\end{aligned}$$

It is easy to verify that all these distance functions are in $[0, 1]$. $\mathrm{Dist}_{\mathrm{Jac}}$ is based on Jaccard coefficient, where the node weights are not taken into account; it is minimized when $S_1 = S_2$, and it equals 1 when their overlap is empty. $\mathrm{Dist}_{\mathrm{Dice}}$ is an extension of the Dice criterion [68], which factors in node weights; $\mathrm{Dist}_{\mathrm{SDice}}$ can be thought of as a scaled version of $\mathrm{Dist}_{\mathrm{Dice}}$: it gives an added premium if the individual weights in S_1 and S_2 are similar. By using min in the numerator, however, we may be penalizing too much for non-equal individual weights, since all that matters is the smaller one rather than some combination of the two. For this reason, $\mathrm{Dist}_{\mathrm{SHel}}$ is an extension of the Hellinger criterion from [68].

[6] These functions were chosen based on their simplicity and naturalness, though other choices are certainly suitable.



(a) Network flow data (b) User query logs

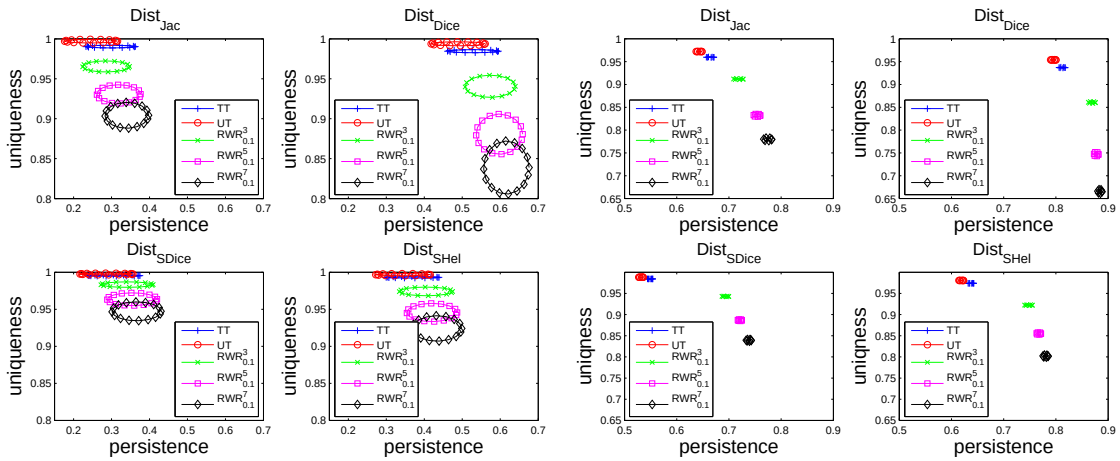
Figure 4.1: Popularity of nodes in signatures.

4.4.3 Experimental Results

To better understand the features selected by the different signature schemes, we first show the number of signatures into which a given node u was selected as part of the signature against u's indegree $|I(u)|$ in Figure 4.1. We show the results for each signature scheme over multiple nodes and both data sets. For one-hop neighbors based approaches (i.e., TT and UT), all points lie under the line $y = x$ because u has only $|I(u)|$ distinct neighbors, and so cannot appear in more signatures for these schemes. The nodes with high x-values from Figure 4.1(a) mostly correspond to popular web sites (e.g. search engines). Thus we were not surprised to see large y-values co-occuring with large x-values for the TT scheme, since edge weights to these popular servers are higher than others. In contrast, UT reaches peak y-values at smaller x-values (between 50 and 100) due to downweighting these popular destinations.

With the RWR scheme, there are numerous points above $y = x$ when both h and c in RWR^h_c are small (e.g., $\text{RWR}^3_{0.1}$)[7], indicating that nodes beyond 1-hop neighborhoods are being selected as part of signatures. With $\text{RWR}^3_{0.1}$, a lot of local hosts include low-indegree nodes in their signatures. This is because a small

[7]When c is as large as 0.9, RWR scheme converges with TT, so we focus on small values of c.



(a) Network flow data (b) User query logs

Figure 4.2: Signature persistence and uniqueness on two real data sets.

h restricts the locality, keeping away from some of the most popular servers. Unlike with TT and UT, RWR^h with a small c-value factors in some "global" information. Therefore, the RWR^h_c scheme (for small c) seems to balance between local and global information. In Figure 4.1(b), we observe a somewhat different point distribution for the user query logs, compared to the network flow data.

Signature persistence and uniqueness. For each t, we summarize the persistence (resp. uniqueness) values using $\mu_p(t)$, $s_p(t)$ — the mean and standard deviation of $\{\text{persistence}_v(t) | v \in V\}$ (resp. $\mu_u(t)$, $s_u(t)$ — the mean and standard deviation of $\{\text{uniqueness}_{v,u} | v, u \in V, v \neq u\}$). We display the span of persistence and uniqueness values as an ellipse: its center is at $(\mu_p(t), \mu_u(t))$; $s_p(t)$ and $s_u(t)$ are the respective (x and y) diameters. Over all different time periods we observed very similar results. Figure 4.2 illustrates results from one time window in depth. We present results from TT, UT and $\text{RWR}^h_{0.1}$ with $h = 3, 5, 7$ and observe that TT lies between UT and $\text{RWR}^h_{0.1}$ in the plots, for both data sets and all distance functions. This is consistent with our intuition that UT downweights universally popular nodes to enhance uniqueness; $\text{RWR}^h_{0.1}$ selects most relevant nodes to i from beyond i's immediate neighborhood to represent it persistently.

(a) AUC from network flow data.

AUC	TT	UT	$\text{RWR}^3_{0.1}$	$\text{RWR}^5_{0.1}$	$\text{RWR}^7_{0.1}$
Dist_{Jac}	0.9086	0.8827	0.9177	0.9087	0.9052
$\text{Dist}_{\text{Dice}}$	0.9093	0.8826	0.9256	0.9172	0.9167
$\text{Dist}_{\text{SDice}}$	0.9035	0.8812	0.9207	0.9086	0.9066
$\text{Dist}_{\text{SHel}}$	0.9094	0.8827	0.9238	0.9162	0.9173

(b) AUC from user query logs.

AUC	TT	UT	$\text{RWR}^3_{0.1}$	$\text{RWR}^5_{0.1}$	$\text{RWR}^7_{0.1}$
Dist_{Jac}	0.9935	0.9969	0.9901	0.9882	0.9877
$\text{Dist}_{\text{Dice}}$	0.9935	0.9969	0.9901	0.9882	0.9877
$\text{Dist}_{\text{SDice}}$	1.0000	1.0000	1.0000	1.0000	1.0000
$\text{Dist}_{\text{SHel}}$	1.0000	1.0000	1.0000	1.0000	1.0000

Table 4.4: Performance on signature persistence and uniqueness

The above figures compare the signature schemes separately in terms of persistence and uniqueness but do not capture the trade-off between the two in a single statistic. For this, we use ROC Curves, a standard measure in statistics [93]. Given G_t and G_{t+1}, for each node v we computed $\text{Dist}(\sigma_t(v), \sigma_{t+1}(u))$ for

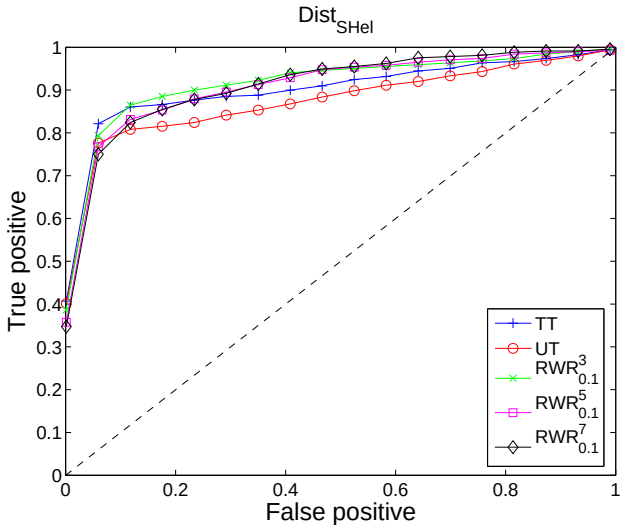


Figure 4.3: ROC curves from network data

all $u \in V$, and returned a ranked list, where u with a smaller Dist-value to v was ranked higher. Our hypothesis is that across time, a node behaves more similar to itself than to others. Therefore in v's ranked list, v should ideally be ranked the first. The ROC curve starts at the origin $(0, 0)$ and traverses the ranked list

of nodes from the top. If the element is v, the ROC curve goes up by a step of 1; otherwise, the ROC curve goes to the right by a step of $1/(|V|-1)$. That is, the x-axis is false positives and y-axis is true positives. We can then compute the Area Under the ROC Curve (AUC). If the AUC is 0.5, the signature scheme is no better than random selection; higher AUC values indicate better accuracy, up to 1 (perfect). We report the average AUC over all v's. Figure 4.3 shows the results on the flow data using $\text{Dist}_{\text{SHel}}$; ROC curves from other distance measures look very similar.

Table 4.3(a) summarizes AUC across different signature schemes per distance measure for the flow data. The multi-hop neighbors based schemes achieved better AUCs than their one-hop counterparts. Among $\text{RWR}^h_{0.1}$ schemes, $\text{RWR}^3_{0.1}$ outperformed the other two. A further observation is that the difference between the AUC from $\text{RWR}^5_{0.1}$ and $\text{RWR}^7_{0.1}$ is small enough to be ignored. Other experiments (not shown) with $\text{RWR}^h_{0.1}$ for $h > 7$ all converged to $\text{RWR}^7_{0.1}$, suggesting that having more than 5 hops does not bring in drastically "new information". This is due in part to the graph having a small diameter: for all h larger than the diameter of the graph, RWR^h coincides with RWR^∞, the unbounded random walk. We repeated the same experiments on user query logs, and summarize AUC values in Table 4.3(b). All signature schemes behave almost equally well on this data set (almost perfectly), with UT being slightly better than the others. In what follows we use $\text{RWR}^3_{0.1}$ as the best representative of the RWR schemes, and do not show results for other parameter settings.

Signature robustness. To evaluate the robustness of the signature schemes, we randomly inserted and deleted edges to obtain a perturbed graph G'_t. Let $\sigma(v)$ and $\hat{\sigma}(v)$ denote v's signatures constructed from G_t and G'_t, respectively. Given a bipartite graph G_t and parameter α, we inserted $\alpha|E_t|$ new edges. First, a node $v' \in V_1$ was sampled proportional to its outdegree, that is, with probability $|O(v')|/\sum_v |O(v)|$. Then a node $u' \in V_2$ was sampled proportional to its indegree,

that is, with probability $|I(u')|/\sum_u |I(u)|$. The weight of (v, u) (initially 0 if edge (v, u) did not previously exist) was assigned independently of $C[v, u]$, but from the total distribution of all edge weights rather than uniformly. For deletions, we sampled existing edges proportional to their edge weights and decremented the weight by one unit, repeating $\beta|E_t|$ times.

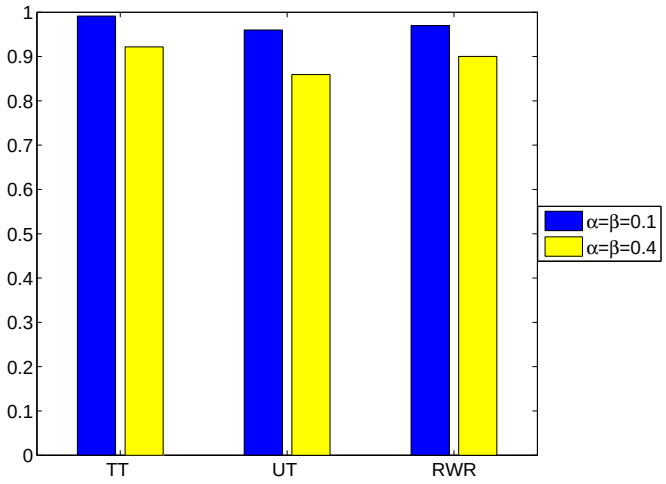


Figure 4.4: Signature Robustness on Network Data

Since our interest in this chapter is in identity matching, we are interested in whether a signature is more similar to its perturbed self than to signatures of other nodes. We again used ROC curves to investigate this and used each $v \in V$ in G_t as a query against V in G'_t, reporting the AUC values in Figure 4.4 for two different parameter settings: $\alpha = \beta = 0.1$ and $\alpha = \beta = 0.4$. TT was the most robust, followed by RWR. UT was the least robust, which is to be expected due to nodes with high indegree (and thus high frequency) being discounted, although the relative difference between all methods is very small.

Summary. The above table summarizes the relative behavior of the signature schemes. We observe an interesting trade off between the three considered schemes: none strictly dominates any other over all three properties. Next we see a clearer separation when applying signatures, which emphasize the properties to differing degrees.

	TT	UT	RWR
persistence	medium	low	high
uniqueness	medium	high	low
robustness	high	low	medium

Table 4.5: Summary of the relative behaviors of the signature schemes

4.5 Application Evaluation

We discuss two applications in detail, and evaluate them empirically on enterprise network flow data.

Multiusage Detection. We follow the discussion of multiusage detection in Section 4.2.4. With network flow data, the problem is to find the set of IPs being multiple connection points (home, office, wireless hotspot) per individual. Our algorithm to detect such an IP set containing v computes the uniqueness values $\text{Dist}(\sigma(v), \sigma(u))$ for all nodes u observed within the same time window. We report those nodes u with low Dist-values (high similarity).

To evaluate the use of signatures for this task, we obtained additional data mapping users to their registered IP addresses, and identified the set of users U who made use of multiple addresses within the enterprise network (of course, this information is not available to the signature-based algorithms). For each user $u \in U$, we denote its set of registered IPs by g_u ($|g_u| > 1$). For the signatures to be effective for this task, our hypothesis is that in one communication graph, signatures for IPs belonging to the same user look more similar to each other, compared to the pairwise similarities between IPs of different users.

For each $v \in g_u$ ($u \in U$), we computed $\text{Dist}(\sigma(v), \sigma(w))$ for all $w \in V$, and derived a ranked list of V sorted by these distances. From these we produced an average ROC curve over all $v \in \bigcup_{u \in U} g_u$, starting at the origin $(0, 0)$. When we traverse the ranked list from the top, if a node is in g_u, the ROC curve goes up by a step of $1/|g_u|$; otherwise, the ROC curve goes to the right by a step

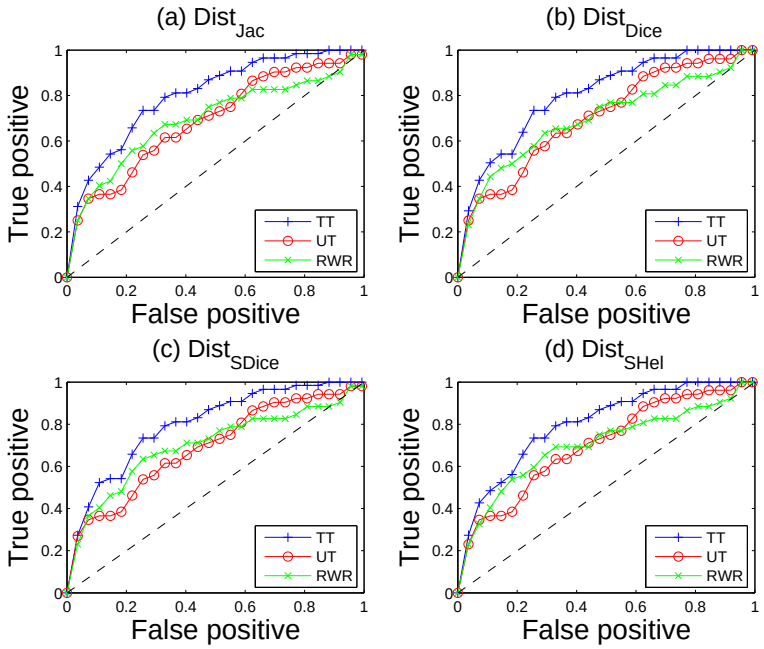


Figure 4.5: Multiusage detection: ROC curves

of $1/|V - g_u|$. So the x-axis measures false positives and y-axis true positives. If the hypothesis is correct, and we can use signatures for this task, then the IPs in g_u should be ranked higher than others. We plot the results across the various schemes in Figure 4.5. Across all distance functions, TT consistently dominates the other two schemes. This agrees with our prediction in Section 4.3 that multiusage detection calls for TT, due to its emphasis on uniqueness and robustness.

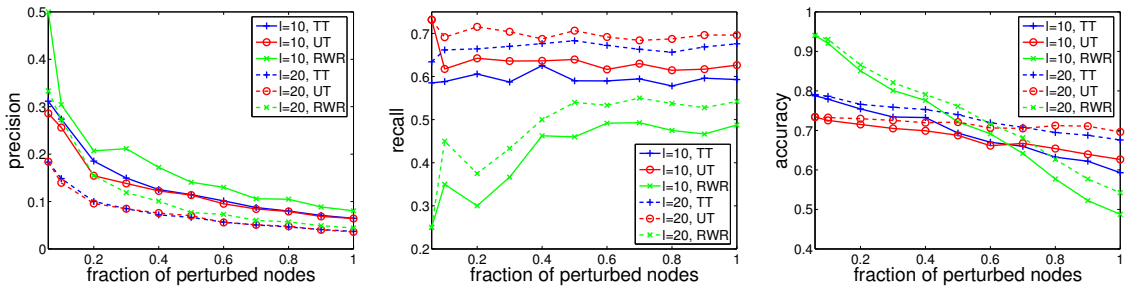
Label Masquerading. For this problem, we simulated masquerading by perturbing $f|V|$ randomly selected nodes (denoted F) in V, where $0 < f < 1$. We created a bijective mapping between nodes in F, and applied this mapping to the communications. We denote the mapping as $E_F = \{(v, u)|v, u \in F\}$, where (v, u) means that v (and all of v's communications) are relabelled with u. Given graphs G_t and G_{t+1} from consecutive time periods, a pair $(v, u) \in E_F$ means that node v in G_{t+1} is relabelled with u, while v's label in G_t remains unchanged. We evaluate our methods on how well they are able to recover E_F.

Algorithm 2 DETECTLABELMASQUERADING(G_t, G_{t+1})

1: Init $V_1 = \emptyset$, $O_F = \emptyset$.
2: **for** each $v \in V$ **do**
3: **if** $1 - \text{Dist}(\sigma_t(v), \sigma_{t+1}(v)) > \delta$ **then**
4: $V_1 = V_1 \cup \{v\}$
5: **else**
6: $\forall u \in V$, $M[u, v] = 1 - \text{Dist}(\sigma_t(v), \sigma_{t+1}(u))$
7: **if** $\exists u \neq v, M[v, u]$ is among v's top-ℓ largest and $M[u, u] \leq \delta$ **then**
8: $O_F = O_F \cup \{(v, u)\}$
9: **else**
10: $V_1 = V_1 \cup \{v\}$
11: **end if**
12: **end if**
13: **end for**

Based on the discussion in Section 4.2.4, the detection algorithm is given in pseudo-code in Algorithm 2. Here V_1 returns the set of local hosts not identified as masqueraders; O_F is the estimate for E_F. We see that an output (v, u) satisfies two conditions: (1) both v and u look different from themselves across time (i.e., low persistence values, from Step 3 and 7 in Algorithm 2); but (2) they look more similar to each other than to others (i.e., high persistence between themselves, from Step 7). We evaluate the various signature schemes for this problem based on the standard information retrieval criteria of precision, recall and accuracy. Here, accuracy measures the percentage of correctly classified hosts, labeled either as "non-suspect" (i.e., $v \notin F$) or with the new label of the node. This combines notions of false positives and false negatives, so we use it as our main evaluation criterion.

In our algorithm, the persistency threshold δ should be a good cutoff between local hosts whose signatures look persistent and those who are not. Therefore, we set δ experimentally as $\delta = \frac{\sum_{v \in V}(1 - \text{Dist}(\sigma_t(v), \sigma_{t+1}(v)))}{p|V|}, p \in \mathbb{N}$. That is, we chose δ as a fraction of the average self-similarity across time (scaled down by p). In particular, we considered $p = 3, 5, 7$ in our experiments, and observed very similar results. Figure 4.6 compares the performance of various schemes with $p = 5$, for various ℓ-values, as a function of fraction of the nodes perturbed.



(a) Precision=$\frac{|O_F \cap E_F|}{|O_F|}$ (b) Recall=$\frac{|O_F \cap E_F|}{|E_F|}$ (c) Accuracy=$\frac{|V_1 \cap (V-F)| + |O_F \cap E_F|}{|V|}$

Figure 4.6: Performance of label masquerading detection.

As expected, recall and accuracy increase as ℓ increases, while precision decreases. Label masquerading should only affect a small fraction of nodes, so we focus our discussion and conclusions on lower values of f. In this range, the RWR scheme outperforms TT and UT in terms of overall accuracy. This coincides with our expectations, since our analysis of this application indicated that label masquerading requires a signature with high persistence and high uniqueness. According to Figure 4.3, which evaluates signature schemes on these measures with network data, RWR is the method of choice.

4.6 Extensions

In general, communication graphs can become extremely large (for example, the graph of all phone calls or internet connections made over the course of a week). Here, we outline some of the scalability issues that arise under such massive data.

Scalable signature computation. When the communication graphs are large, even storing the graph can become infeasible. Instead we need compact data structures that can process an observed sequence of communications (defining edges in the graph), from which we can extract (approximate) signatures. Our assumption is that although the total volume of edges is too massive, we can at least record some constant amount of information about each node in turn:

this is the "semi-streaming" model of graph stream processing [96]. Any given signature scheme will need a different approach in this restricted model, here we outline methods for the signature schemes used here for illustration. For the Top Talkers, we need to find the approximate heaviest-weight neighbors of node i. If the communication is pre-aggregated, we can just keep a heap of heavy edges; but more realistically, we see each individual communication, and want to recover the most frequent. We can use summary structures such as a CM sketch for each node to find its heaviest outgoing edges, and hence its signature [30]. For Unexpected Talkers, the situation is more complex. Here, we can additionally keep an FM sketch for each node, to find its incoming degree [51]. To find the signature for a node i, we can use the CM sketch to estimate $C[i, j]$, and the FM sketch to estimate $|I(j)|$ for each node j; combining these gives an approximation of $C[i, j]/|I(j)|$ as required. For schemes based on Random Walk with Reset, there is less prior work to draw on. Techniques in [107] give approaches to make the comptuations more scalable, based on appropriate block-wise decompositions of the graph; extending these to the full semi-streaming model remains an open problem.

Scalable signature comparison. When there are large number of nodes with signatures, applications based on comparing many signatures together become expensive (potentially quadratic in the number of nodes). At the heart of many applications discussed above is the problem of, given a signature, finding the most similar signature(s) from a particular (sub)set. This fits the set up of the nearest neighbor problem. Even for moderately small signature sizes, this can become expensive, and so we can turn to approximate nearest neighbor algorithms. Here, different approaches are needed for each different distance function, rather than signature scheme. For example, efficient solutions exist where the distance function is the Jacard distance, by using an approach based on Locality Sensitive Hashing [75].

4.7 Related Work

Signatures (and fingerprints) were classically studied as an application of statistical pattern recognition, an area of study with a long history of techniques for feature selection and classification [77]. However, these generic techniques are less appropriate given contextual information as in the case of communication graphs.

Usage profiling in graphs (and networks) has been studied in multiple settings [99, 117, 85, 107, 73, 98]. However, the goal of these works is to model behavior aggregated at the level of the entire graph to detect network-wide anomalies. Usage profiling at the granularity of the *individual* has been studied for activity monitoring [48], for user recognition [106], and for enterprise security [83]. The approach uses complex rules on records of individual activity requiring detailed data storage.

Cortes *et al.* initiated the study of "COI-based" signatures, which makes use of communication graph topology to design individual's signatures in a concise way for detecting fraudulent users [35, 36], identifying repetitive debtors [68] and predicting links for viral marketing [70]; similar techniques were also applied to identify authors from bibliographical citations in [69]. However, they only focused on particular signatures and their applicability to the motivating tasks. We do not know of any prior work that proposed a principled approach such as ours for a detailed classification of properties of signatures and studying applications based on what properties they need for signatures to be useful.

4.8 Chapter Summary

Nowadays interaction or communication between individuals is ubiquitous. Given this abundance of communication, many applications emerge which rely on analyzing the patterns behind the communications. The idea of using signatures to capture behavior is an attractive one. Here, we have attempted to take a very

general approach to problems of defining and analyzing signatures in communication graphs. We proposed a general framework for understanding and analyzing these signatures, based on the three fundamental and natural properties of persistence, uniqueness and robustness. We justified these properties by showing how they impact a broad set of applications. We explored several signature schemes in our framework and evaluated them on real data in terms of these properties. In particular, our study on two concrete applications using signatures demonstrates their effectiveness on real data experimentally. This study underlined the fact that there is not one single signature scheme which is good for all applications, but rather that different signatures are needed, depending on what balance of the three properties they provide. We believe that a larger suite of properties of signatures are needed for the space of all applications that signatures will be useful for.

Chapter 5

Modeling Distributional Changes via Sequential Probability Ratio Test

5.1 Introduction

Model-based declarative queries are becoming an attractive paradigm for interacting with many data stream applications. This has led to the development of techniques to accurately answer the queries using distributional models rather than raw values. The quintessential problem with this is that of detecting when there is a change in the input stream [53, 72, 111], which makes models stale and inaccurate. In IP network management, changes in the traffic patterns might indicate an intrusion, attack or an anomaly that needs attention of the network managers [79, 84, 31]. Likewise in financial streams, a change in the pattern of trades might represent an opportunity or a warning for the analyst.

In particular, changes may be global, involving the entire distribution of items (such as determining if the distributions are far apart in some measure), or local, involving the distribution of singleton items (such as, if individual probability differs significantly in one distribution to another).

Change detection between two stored data sets in general brings up fundamental questions such as how to measure change and what is a threshold for change. In the context of data streams, additional questions arise. In particular, what are the "two" data sets that would be basis for detecting a change? Typically, data stream queries specify some window W on which the query is executed. For example, there are tumbling window queries or sliding window queries [96, 9]. An

issue is how to determine the W. Although the need of continuous detection for changes gives rise to the sliding window model [80, 39], in change detection problems, fixing a window size can only work well if information on the time-scale of change is available, but this is rarely the case. Indeed, we need scale-free detection algorithm to find changes as quickly as possible. However, this goal may be delayed by the size of the window, since a window smaller than the changing rate may miss the detection or capture the change at a later time when gradual changes cumulate over time, and a window larger than the changing rate delays the detection until the end of the window. One can tradeoff some performance by searching over multiple sliding windows (of different sizes) in parallel by spending some analysis time and space [80, 39]. But it is infeasible to exhaustively search over all possible window sizes. A different strategy uses a decaying-weighted window [84, 40] to weight the importance of examples according to their age. In this case, the choice of a decay constant should match the unknown rate of change. Still, what is needed is a seamless change detection method, such that whenever a change occurs, it accurately estimates when it happens with short detection delay.

In this chapter, we present such change detection algorithms that are inspired by the well-known *sequential probability ratio test* [109, 14] in Statistics. It comes with sound basis for setting the threshold for change, for providing guarantees on keeping track of the change point incrementally and for the minimum delay in detecting a change, under the assumptions that (1) distributions before and after a change are both known as *a priori*; (2) observations are generated independently.

Our algorithmic contribution is to design a very space-efficient, yet fast method for change detection in one pass over the data streams, where we have to relax the above two assumptions. Furthermore, we show that it not only holds for global changes in the distributions as is studied in Statistics, but also holds for local changes that have not been addressed earlier. We perform a thorough study of

our algorithms in practice with synthetic and real data sets. We not only show the accuracy and robustness of our algorithms to detect changes in one pass, but also demonstrate the accurate change-point estimate, relative to those obtained offline. Our approaches have short detection delay, compared to alternative window-based solutions. Finally, we study real traces with web logs to demonstrate how an accurate change-point estimate can help improve model-based query qualities. Another case study with labelled attacks shows experimental evidence that our methods have the potential power for intrusion detection monitoring. Still, better understanding of how attacks incur changes is needed to infer critical things.

The chapter is organized as follows. We define both global and local change detection problems in Section 5.2. We present our global change detection algorithms in Section 5.3 and present experimental studies in Section 5.5. Likewise, we present algorithmic methods and experimental studies in Section 5.4 and 5.6 respectively for local change detection. Related work is in Section 5.7, with conclusions in Section 5.8.

5.2 Change Detection Problems

$A_1^n = (x_1 \dots x_n)$ is a sequence of input items, with $x_i \in U = \{0..u-1\}$, generated from some underlying distribution at time i. Its sub-sequence $A_1^w = (x_1 \dots x_w)$, $w < n$, is a baseline data set generated from an original distribution P_0. With P_0 known as *a priori*, two situations are possible: either every $x_i \in A_1^n$ is generated from P_0, or there exists an *unknown change point* $w < \lambda \leq n$ such that, $x_i \sim P_0$ for $i < \lambda$ and $x_i \sim P_1$ for $w < \lambda \leq i \leq n$, where P_0 and P_1 are called pre- and post-change distributions. We say that a change in P_0 has occurred if P_1 differs significantly from P_0. By significance we mean the amount of change, measured by a distance function $D_\lambda(P_1||P_0)$, is greater than some threshold. Here $D_\lambda(P_1||P_0)$ depends on the change point λ.

A change detection problem is based on hypotheses: $\mathcal{H}_0$ — there is no change in P_0; and $\mathcal{H}_1$, otherwise. Here $\mathcal{H}_1$ is a composite of a set of "parallel" tests $\{H_\lambda | w < \lambda \leq n\}$, where H_λ is the hypothesis that a change occurs at time λ; that is, $\mathcal{H}_1 = \bigcup_{w<\lambda\leq n} H_\lambda$. In one pass over the data streams, data is presented one point at a time. To quickly detect a change after its occurrence, the best opportunity is to make a declaration after each observation, in the process of data acquisition. Therefore at each x_n, a desired change detection algorithm operates in two phases: it first locates an index $w < \lambda \leq n$, which has the highest probability of being a change point (i.e., the largest $D_\lambda(P_1||P_0)$); in the second phase, we conduct a hypothesis testing to accept or reject λ as a change point (i.e. to accept or reject $\mathcal{H}_1$). To interpret the change, the hypothesis testing compares the largest $D_\lambda(P_1||P_0)$ against a threshold to tell whether the difference is significant. Once we accept H_λ at time n, we call λ the *change point*, and n the *change detection point*; otherwise, we continue to make an additional observation. Formally,

Definition 6 *Let $(x_1, ..., x_n)$ be an input stream, and its sub-stream $(x_1, ..., x_w)$, $w < n$, is generated from a prior distribution P_0. A change detection problem is to decide whether to accept $\mathcal{H}_1$ (i.e., reject $\mathcal{H}_0$) or not. We accept $\mathcal{H}_1$ at time n when*

- *A global change in entire distribution is detected when*

$$T^n = D_{\lambda^*}(P_1||P_0) = \max_{w<\lambda\leq n} D_\lambda(P_1||P_0) \geq \tau_g,$$

- *A local change in distribution of a singleton item $j \in U$ is detected when*

$$T^n = D_{\lambda^*}(P_1[j]||P_0[j]) = \max_{w<\lambda\leq n} D_\lambda(P_1[j]||P_0[j]) \geq \tau_l,$$

where τ_g and τ_l are global and local change detection threshold to be specified in Section 5.3.1 and 5.4. A change detection algorithm outputs: $\lambda^* = \arg\max_{w<\lambda\leq n} D_\lambda(P_1||P_0)$

as a change-point estimate for a global change; $\lambda^* = \arg\max_{w<\lambda\leq n} D_\lambda(P_1[j]||P_0[j])$ *as a change-point estimate for a local change at item* j*, where* P_0 *and* P_1 *are pre- and post-change distributions;* $P_q[j]$ *denotes the probability of seeing* j *under* P_q*,* $q = 0, 1$.

Global and local change detection problems are fundamentally different: global change detection finds when the entire distribution P_1 differs significantly from P_0; local change detection finds where inside distribution $P_1[j]$ and $P_0[j]$ (i.e., the distribution of a singleton item) differ significantly. Depending on the nature of the change, for some cases the local change detection methods succeed and for some, the global methods succeed in catching the changes (see examples in Figure 5.1). In real applications, we are equally interested in detecting local and global changes. Therefore, we formulate them as two separate problems.

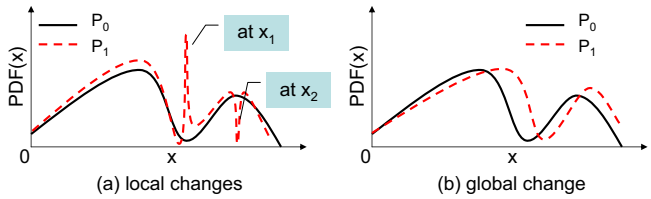


Figure 5.1: Examples of local and global changes. P_0 and P_1 are respective probability density function (or PDF) of pre- and post-change distributions.

5.3 Our Global Change Detection Algorithms

5.3.1 Preliminary

Sequential change detection algorithms base their detection decision on the classic *Sequential Probability Ratio Test (SPRT)* [109]. When the underlying distribution changes from P_0 to P_1 at point λ, it is natural that the probability of observing sub-sequence $A_\lambda^n = (x_\lambda, ..., x_n)$ under P_1 is "significantly" higher than that under P_0. By significance, we mean the ratio of the two probabilities is no smaller

than a threshold. Given that both P_0 and P_1 are known as *a priori*, and that points in the stream are independently generated, the test statistic for testing the hypothesis H_λ that a change occurrs at time λ against $\mathcal{H}_0$ that there is no change is equal to

$$\begin{aligned} T_\lambda^n &= D_\lambda(P_1||P_0) = \log \frac{\Pr(x_\lambda ... x_n|P_1)}{\Pr(x_\lambda ... x_n|P_0)} \\ &= \sum_{i=\lambda}^{n} \log \frac{P_1[x_i]}{P_0[x_i]} = T_\lambda^{n-1} + \log \frac{P_1[x_n]}{P_0[x_n]}. \end{aligned}$$

Page's (CUSUM) procedure [14] identifies that the statistic $T^n = \max_{w<\lambda\leq n} T_\lambda^n$ obeys the recursion ($T^0 = 0$):

$$T^n = \max\left(0, T^{n-1} + \log \frac{P_1[x_n]}{P_0[x_n]}\right).$$

Therefore, it takes each new item $O(1)$ time to update T^n and to detect a global change by testing $T^n \geq \tau_g$. Accordingly, we update the most likely change point to n in $O(1)$ time when $T^n = 0$, without extra overhead. When a change is detected at time n, the change-point estimate is $\lambda^* = \max\{\lambda | T^\lambda = 0, w < \lambda \leq n\}$. Therefore, the adaptive window size at time n is $w^{(n)} = w^{(n-1)} \cdot 1_{\{T^n>0\}} + 1$, where $1_{\{x\}}$ is the indicator of event x, $w^{(n)}$ is the number of observations after the last time the test statistic T^n is reset to zero, and the start point of the so-called adaptive window estimates the change point. In addition, Wald's approximation [109] says that global threshold $\tau_g = \log((1-\beta)/\alpha)$, given user-specified false alarm rate $\alpha = \Pr(\mathcal{H}_1|\mathcal{H}_0) = \Pr(T^n \geq \tau_g|\mathcal{H}_0)$ and miss detection rate $\beta = \Pr(\mathcal{H}_0|\mathcal{H}_1) = \Pr(T^n < \tau_g|\mathcal{H}_1)$.

Here T^n is an integration of the log likelihood ratios of the most recent $(n - \lambda + 1)$ observations, such that T_λ^n is maximized. Thus, it is inherent that only items after the change point contribute to T^n. And the natural advantage of this statistic is to detect changes at different scales without instantiating windows of different sizes in parallel.

5.3.2 Our Offline Algorithm

All discussions in Section 5.3.1 assume that both P_0 and P_1 are known as *a priori*. In practice, it is fair to assume the availability of a "base" distribution P_0, but not the prior knowledge of P_1. So (1) how to derive the post-change distribution P_1 from the stream? How to incrementally maintain it? (2) What if observations in stream are not independent, just as with streaming data? We shall address these issues in this Section. For streaming concerns, we design a fast and small-space algorithm in Section 5.3.3.

In the context of our problem, the input is a sequence of points $A_1^n = (x_1 \dots x_w \dots x_n)$; each $x_i \in U$. The original distribution P_0 is known as *a priori*, and is constructed from the first w observations $A_1^w = (x_1 \dots x_w)$. It is defined as a vector representing the relative frequency of each distinct element $j \in U$, with smoothing:

$$P_0[j] = \frac{|\{i | x_i = j, 1 \leq i \leq w\}| + \gamma}{w + \gamma \cdot u} = \frac{S_1^w[j] + \gamma}{w + \gamma \cdot u}.$$

Here $S_1^w[j]$ denotes the frequency for item j in A_1^w. γ is usually set to 0.5 for smoothing, which controls the sensitivity to previously unseen items (i.e., $\forall j \in U,\ P_0[j] > 0$); $u = |U|$ is the domain size. We assume that the size w of the baseline data set is large enough to guarantee the goodness of P_0, compared to its true counterpart. There are a variety of methods to capture the underlying distribution of a sequence. For simplicity and popularity reasons, we use empirical distributions after smoothing throughout the chapter. However, our change detection methodology, as a whole, fits well with other alternatives. That is, we can isolate definition of a change and mechanism to detect it from the data representation itself.

Our P_1 is incrementally updated based on P_0 with new arrivals; that is at any time n, P_1^n is derived from A_1^n: $\forall j \in U,\ P_1^n[j] = \frac{S_1^n[j]+\gamma}{n+\gamma \cdot u}$. An inherent problem with this setup is that when a change occurs, P_1 constructed based on all observations A_1^n makes use of stale data, thus is reluctant to reflect the change

promptly. Our argument is that having a change-point-dependent test statistic greatly alleviates such detection delays. We will study the effectiveness of this setup against alternative solutions in experiments. In practice, we can improve our algorithm by adopting, for example sliding window [40] or decaying window [27] models, to build more up-to-date P_1. We consider such variants orthogonal to our main line of inquiry, and do not consider it explicitly any further.

Theorem 7 *With dependent observations,* $T_\lambda^n \simeq Z_\lambda^n = \sum_{i=\lambda}^n \log \frac{P_1^i[x_i]}{P_0[x_i]}$, *where* $P_1^i[x_i] = \frac{S_1^i[x_i]+\gamma}{i+\gamma \cdot u}$; $S_1^i[x_i] = |\{j | x_j = x_i, 1 \leq j \leq i\}|$; *the smoothing factor* $\gamma = 0.5$; $u = |U|$.

Proof. According to the definition for T_λ^n,

$$\begin{aligned} T_\lambda^n &= \log \frac{\Pr(x_\lambda ... x_n | P_1)}{\Pr(x_\lambda ... x_n | P_0)} \\ &= \log \frac{\Pr(x_n | x_\lambda ... x_{n-1}, P_1) \cdot \Pr(x_\lambda ... x_{n-1} | P_1)}{\Pr(x_n | x_\lambda ... x_{n-1}, P_0) \cdot \Pr(x_\lambda ... x_{n-1} | P_0)} \\ &\simeq \log \frac{\prod_{i=\lambda}^n P_1^i[x_i]}{\prod_{i=\lambda}^n P_0[x_i]} = \sum_{i=\lambda}^n \log \frac{P_1^i[x_i]}{P_0[x_i]} \\ &\equiv Z_\lambda^n = Z_\lambda^{n-1} + \log \frac{P_1^n[x_n]}{P_0[x_n]}. \end{aligned}$$

The second equality holds due to Bayes's rule; "$\simeq$" means asymptotically equal since $i \geq \lambda > w \gg 0$, and holds according to the definition for P_1^i and the fact that observations $x_\lambda \dots x_n$ are independent under *a priori* P_0. ∎

The proof shows that the incremental maintenance of test statistics is by no means restricted to the independence assumption. It is generally true even for dependent observations. Again, test statistic $Z^n = \max_{w<\lambda\leq n} Z_\lambda^n$ for our offline global change detection algorithm obeys the recursion ($Z^0 = 0$):

$$Z^n = \max\left(0, Z^{n-1} + \log \frac{P_1^n[x_n]}{P_0[x_n]}\right).$$

Figure 5.2 illustrates the basic structure of the algorithm: "base" distribution P_0 is derived from $(x_1 \dots x_w)$ as *a priori*; P_1^n is the empirical distribution over

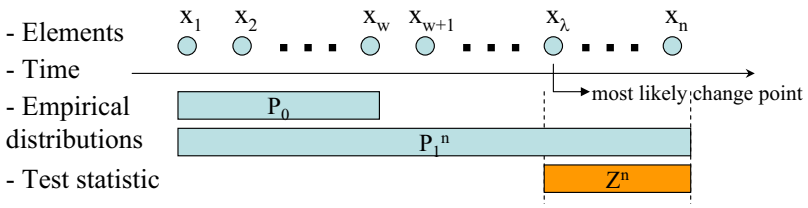


Figure 5.2: An illustration for the sequential change detection algorithm.

$(x_1 \dots x_n)$; test statistic Z^n integrates the change-point estimate and is evaluated incrementally only according to points after the change. Formally, at any time n:

1. It takes $\Theta(1)$ time to update $S_1^n[x_n] \leftarrow S_1^{n-1}[x_n] + 1$ and $P_1^n[x_n]$.

2. It takes $\Theta(1)$ time to update test statistic Z^n according to the recursion. If $Z^n \geq \tau_g$, report a detected global change and the associated most likely change point λ^*. Otherwise, if $Z^n = 0$, update $\lambda^* = n$.

Theorem 8 *Our offline sequential change detection algorithm takes $O(u)$ space and $O(1)$ per-item processing time to detect a global change, where $u = |U|$.*

Proof. It takes $O(u)$ space to store S_1^w and S_1^n exactly, plus another two words of memory for respective Z^n and λ^*. ∎

5.3.3 Streaming Algorithm

We will use previously known *sketches* [96, 9] as basic components to summarize the input in small space. Then we will estimate test statistic Z^n and most likely change point λ^* from such small-space structures. One simple attempt is to create sketches for S_1^w and S_1^n, hence to derive good estimates for P_0 and P_1^n, respectively. For every new item x_n, the estimate for test statistic Z^n is updated through $\hat{Z}^n = \max\left(0, \hat{Z}^{n-1} + \log \frac{\hat{P_1}^n[x_n]}{\hat{P_0}[x_n]}\right)$. However, this method is severely flawed in two ways. First, approximating the ratio of values in small space could perform poorly, although we have accurate estimates for values. Second, even though

we obtain accurate estimate per log-ratio, the additive errors accumulated with infinite new items might lead to unbounded errors. Therefore, a more careful design of a streaming algorithm is desired.

High-level ideas. Since the simple combination of incremental updates (via recursion) and sketches fails to accurately detect changes, we focus on designing a sketch-based algorithm to estimate test statistic $\hat{Z}^n$ *directly* in polylogarithmic time per item. Although this is slightly weaker, compared to the offline algorithm's $O(1)$ per-item processing time, we have immediate gain in space. Unlike with the offline solution, we do not incrementally update $\hat{Z}^n$, neither does $\hat{\lambda}^*$. Thus, we will derive $\hat{\lambda}^*$ based on multiple $\hat{Z}^n_\lambda$ estimates. Instead of exhausting all $w < \lambda \leq n$ (i.e., $O(n)$) points and taking the max, we probe at values of the change point λ that form a geometric progression. For example, we may estimate $\hat{Z}^n_\lambda$ only for $\lambda = n, n-1, n-2, n-4, \ldots, n-2^i, \ldots$. The justification is that it achieves a dramatic reduction in computation time, since we need only $O(\log n)$ estimates to keep track of. Our method will give good accuracy for λ closer to n, exactly because for such λ's we have many points to interpolate; it may give a larger error for $\lambda \ll n$, but the relative error will probably be small.

Data structures. In order to obtain a good estimate for test statistics in form of log likelihood ratios, we need to pre-process $A^w_1 = (x_1 \ldots x_w)$ to be aggregated to $S^w_1[i] = |j|x_j = i, 1 \leq j \leq w|$. And this is feasible due to our prior knowledge on P_0. As for streaming algorithm, let S_{1/P_0} be the stream whose ith entry is $S_{1/P_0}[i] = \frac{1}{P_0[i]} = \frac{w+\gamma \cdot u}{S^w_1[i]+\gamma}$. We keep a CM sketch (see Section 2.2.1) for this "inverted" stream, denoted CM_{1/P_0}. It can be shown that $\forall i \in U$, $\mathrm{CM}_{1/P_0}[i] = \min_j count[j, h_j(i)]$ is a good estimate for $1/P_0[i]$.

On the other hand, we will use EH+CM to summarize $A^n_1 = (x_1 \ldots x_n)$, such that we can use it to answer point queries $S^n_\lambda[i]$, for any $1 \leq \lambda \leq n$, with accuracy guarantees. We refer to this structure by EHCM_{S_1}. EH+CM is an example of *cascaded summary*, or a "sketch within a sketch": each bucket in

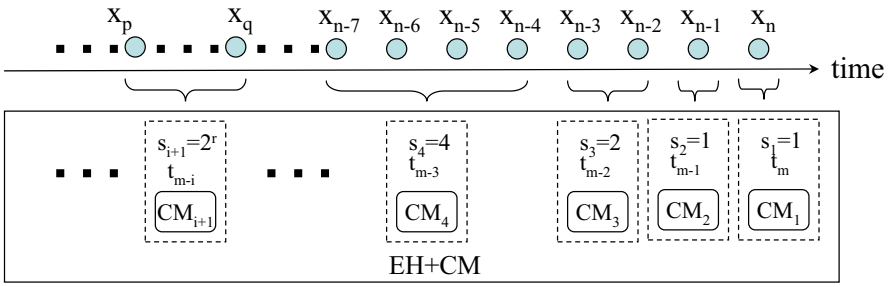


Figure 5.3: Sketch structure for EH+CM.

EHs (see Section 2.2.3) contains a CM sketch to summarize data points seen in a time range $t \in (t_{i-1}, t_i]$. Associated with this CM sketch is its size s (i.e., the number of items the CM summarizes) and the timestamp t_i. The data structure is illustrated graphically in Figure 5.3. Each CM sketch we use takes the same set of hash functions, with $(\epsilon, \delta/\log n)$-guarantees (i.e., sketch width is e/ϵ; depth is $\log \frac{\log n}{\delta}$), so they are linearly summable.

Lemma 1 *The space overhead of the EH+CM structure is* $O(\frac{\log n}{\epsilon} \log \frac{\log n}{\delta})$.

Proof. There are $O(\log n)$ buckets; each takes $O(\frac{1}{\epsilon} \log \frac{\log n}{\delta})$ space for a CM sketch with $(\epsilon, \delta/\log n)$-guarantees, due to Fact 1. Then applying the union bound ensures that, after summing up $O(\log n)$ CM sketches in EH+CM, the probability of failure for each query is still bounded by δ. ∎

Updates. $\text{CM}_{1/P_0}$is constructed during pre-processing. So we only update EHCM_{S_1}on live data as it arrives:

1. For each new item, create a new bucket with size 1 and the current timestamp, and initialize a CM sketch with the single item.

2. Traverse the list of buckets in order of increasing sizes (right to left). If $s_{q+2} + s_{q+1} \leq \epsilon \sum_{i=1}^{q} s_i$, merge the $(q+1)$th and the $(q+2)$th buckets into a single bucket. The timestamp for this new bucket is that of the $(q+1)$th bucket; its CM sketch is the linear combination of CM_{q+1} and

CM_{q+2}: $count'_{q+1}[i,j] = count_{q+1}[i,j] + count_{q+2}[i,j]$. We may need to do more than one merge.

Lemma 2 *To update the EHCM$_{S_1}$, the per-item processing time is $O(\frac{\log n}{\epsilon}\log\frac{\log n}{\delta})$ in the worst case, and its amortized time is $O(\frac{1}{\epsilon}\log\frac{\log n}{\delta})$.*

Proof. Linearly combine two CM sketches takes $O(\frac{1}{\epsilon}\log\frac{\log n}{\delta})$ time. And the arrival of a new element can be processed by one merger of buckets on average and $O(\log n)$ mergers in the worst case (see [40]). ∎

Change detection. Assume that at time n, we have m buckets whose timestamps are after w, $B_1 \ldots B_m$, ordered from the most- to the least-recent; each $B_i = (s_i, t_{m-i+1}, \mathrm{CM}_i)$. The global change detection algorithm consists of two components. For each new item x_n:

1. Estimate $\hat{Z}^n_\lambda$ directly from sketches, for $\lambda = t_1{+}1, \ldots, t_{m-1}{+}1$. The intuition comes from the fact that $T^n_\lambda = \sum_{i=\lambda}^n \log\frac{P_1[x_i]}{P_0[x_i]} = \sum_{j\in U}\left(S^n_\lambda[j]\cdot\log\frac{P_1[j]}{P_0[j]}\right)$, where $S^n_\lambda[j] = |\{i|x_i = j, \lambda \le i \le n\}|$. We first derive a CM sketch for P^n_1, denoted $\mathrm{CM}_{P^n_1}$, by linearly combining all the sketches in EHCM_{S_1} and updating every entry in the resulting table: $count_{\mathrm{CM}_{P^n_1}}[i,j] \leftarrow \frac{^{count}\mathrm{CM}_{P^n_1}{}^{[i,j]+\gamma}}{n+\gamma\cdot u}$. Next, we build a sketch $\mathrm{CM}_{\log P^n_1/P_0} = \log(\mathrm{CM}_{P^n_1} \times \mathrm{CM}_{1/P_0})$, where every entry in the table has: $count_{\mathrm{CM}_{\log P^n_1/P_0}}[i,j] = \log(count_{\mathrm{CM}_{P^n_1}}[i,j]\cdot count_{\mathrm{CM}_{1/P_0}}[i,j])$. Then for each $\lambda = t_q + 1$ $(q = 1..m-1)$, we compute CM^n_λ by the linear combination of sketches: $\mathrm{CM}^n_\lambda = \sum_{j=q+1}^m \mathrm{CM}_j$. This gives good estimate for S^n_λ. Finally,

$$\hat{Z}^n_\lambda = \mathrm{median}_{i=1}^{\log(\log n/\delta)} \sum_{j=1}^{e/\epsilon}(count_{\mathrm{CM}^n_\lambda}[i,j]\cdot count_{\mathrm{CM}_{\log P^n_1/P_0}}[i,j]). \quad (5.1)$$

2. Change-point estimate. Having $\hat{Z}^n_\lambda$ for $\lambda = t_1 + 1 \ldots t_{m-1} + 1$, the missing estimates for $\hat{Z}^n_\lambda$ for other λ-values $(w < \lambda \le n)$ are approximated by interpolation with a cubic spline. After interpolation, we can use any known

method to find the local maxima. If the local maximum is larger than the threshold τ_g, we report a global change, and the maximizer is the change-point estimate.

Lemma 3 *Our streaming change detection procedure takes* $O(\frac{\log n}{\epsilon}\log\frac{\log n}{\delta})$ *per-item processing time to detect a global change, and to estimate the change point when it occurs.*

Proof. Given n, it takes $O(\frac{\log n}{\epsilon}\log\frac{\log n}{\delta})$ time to construct $\mathrm{CM}_{P_1^n}$, and $O(\frac{1}{\epsilon}\log\frac{\log n}{\delta})$ time to build $\mathrm{CM}_{\log P_1^n/P_0}$. Then for each $\lambda = t_q + 1$ ($q = 1..m-1$), incrementally computing CM_λ^n: $\mathrm{CM}_{t_q+1}^n = \mathrm{CM}_{t_{q+1}+1}^n + \mathrm{CM}_{q+1}^n$ takes $O(\frac{1}{\epsilon}\log\frac{\log n}{\delta})$ time, and then the estimate for $\hat{Z}_\lambda^n$ is output in $O(\frac{1}{\epsilon}\log\frac{\log n}{\delta})$ time. We have in total $m = O(\log n)$ [40] $\hat{Z}_\lambda^n$ to estimate, hence, the overall time complexity is $O(\frac{\log n}{\epsilon}\log\frac{\log n}{\delta})$. ∎

5.4 Our Local Change Detection Algorithms

Offline Algorithm. Local change detection problem detects change in distribution $P_0[j]$ of any singleton item $j \in U$. Recall that $A_s^n = (x_s \dots x_n)$. To make our notation consistent with that for global changes, for each distinct item value $j \in U$, we define $A_{j,s}^n = (i_{j_1}, i_{j_2}, ..., i_{j_{n_j}})$ to be a sequence of timestamps when item j is observed in A_s^n, i.e., $\forall k \in \{j_1, ..., j_{n_j}\}$, $s = i_{j_1} \le i_k \le n$, $x_{i_k} = j$, and $n_j = |\{i | x_i = j, s \le i \le n\}|$ records the frequency of j in A_s^n. For example, given $A_1^n = (1,3,2,4,2,3)$, $n = 6$, when $s = 2$, $j = 3$, $A_{j,s}^n = (2,6)$, since $x_2 = x_6 = 3$. When the underlying distribution $P_0[j]$ to generate item j changes to $P_1[j]$ at λ ($\lambda \in A_{j,w+1}^n$ and $x_\lambda = j$), it is more likely to observe $A_{j,\lambda}^n$ under $P_1[j]$ than $P_0[j]$. Similar to the distance function for the global change detection method, we have

Definition 7 *Given sequence* $(x_1, ..., x_\lambda, ..., x_n)$, λ *is a change point of the singleton* j, $P_0[j]$ *and* $P_1[j]$ *are its pre- and post-change distributions, then the local*

distance between $P_0[j]$ *and* $P_1[j]$ *is*

$$T^n_{j,\lambda} = D_\lambda(P_1[j]||P_0[j]) = \log \frac{Pr(A^n_{j,\lambda}|P_1[j])}{Pr(A^n_{j,\lambda}|P_0[j])},$$

where $Pr(A^n_{j,\lambda}|P_i[j])$ *is the probability of observing sequence of timestamps* $A^n_{j,\lambda}$ *under* $P_i[j]$, $i = 0, 1$, $j \in U$; $x_\lambda = j$.

According to Theorem 7, with dependent observations, and when post-change distribution is not known as *a priori*, test statistic for a local change hypothesis H_λ at singleton j is

$$Z^n_{j,\lambda} = \sum_{i=\lambda}^{n} \left(1_{\{x_i=j\}} \cdot \log \frac{P^i_1[x_i]}{P_0[x_i]} \right) = Z^{n-1}_{j,\lambda} + 1_{\{x_n=j\}} \cdot \log \frac{P^n_1[x_n]}{P_0[x_n]},$$

where $1_{\{x\}}$ is the indicator of event x; $P^i_1[x_i] = \frac{S^i_1[x_i]+\gamma}{i+\gamma \cdot u}$; $S^i_1[x_i] = |\{j|x_j = x_i, 1 \leq j \leq i\}|$; the smoothing factor $\gamma = 0.5$; $u = |U|$. Again, test statistic $Z^n_j = \max_{w<\lambda\leq n} Z^n_{j,\lambda}$ for a local change at singleton j obeys the recursion ($Z^n_j = 0$):

$$Z^n_j = \max \left(0, Z^{n-1}_j + 1_{\{x_n=j\}} \cdot \log \frac{P^n_1[x_n]}{P_0[x_n]} \right). \tag{5.2}$$

Theorem 9 *Given false alarm probability* α *and miss detection probability* β, *local change detection threshold* $\tau_l = \log((1-\beta)/\alpha)$.

Proof. For simplicity, we drop off change point in this proof. Consider a sample path of timestamps $A_j = (i_{j_1}, i_{j_2}, ..., i_{j_{n_j}})$, where item j is observed at time $i_{j_k}, k = 1..n_j$, and on the n_j-th observation of j the threshold τ_l is hit and a change in $P_0[j]$ is detected (i.e. when $\mathcal{H}_1$ is accepted). Thus:

$$D(P_1[j]||P_0[j]) = \log \frac{\Pr(A_j|\mathcal{H}_1)}{\Pr(A_j|\mathcal{H}_0)} \geq \tau_l.$$

For any such sample path, the probability $\Pr(A_j|\mathcal{H}_1)$ is at least 2^{τ_l} times as large as $\Pr(A_j|\mathcal{H}_0)$, and this is true for all sample paths where the test terminates with $\mathcal{H}_1$ being accepted. Hence, the probability measure of the totality of all sample paths where $\mathcal{H}_1$ is accepted when $\mathcal{H}_1$ is true is at least 2^{τ_l} times the probability

measure of the totality of all sample paths where $\mathcal{H}_1$ is accepted when $\mathcal{H}_0$ is true. The former probability (accept $\mathcal{H}_1$ when $\mathcal{H}_1$ is true) is $1-\beta$; the latter probability (accept $\mathcal{H}_1$ when $\mathcal{H}_0$ is true) is equal to α. This gives an upper bound on threshold: $\tau_l \leq \log((1-\beta)/\alpha)$. In practice, we use $\log((1-\beta)/\alpha)$ as an approximation for τ_l. ∎

It is clear that this proof is distribution free.

Put everything together, at any timestamp n, test statistics for local change detections can be updated in $O(1)$ time, using recurrence in Eq. (5.2). In particular, test statistic Z_j^n for a local change in $P_0[j]$ is updated by the current observation only when $x_n = j$. Then we conduct a hypothesis test that compares Z_j^n against the threshold τ_l for local changes. If $Z_j^n \geq \tau_l$, we say that a local change in $P_0[j]$ is detected at time n, and the change-point estimate is $\lambda_j^* = \max\{\lambda | Z_j^\lambda = 0, w < \lambda \leq n\}$.

Corollary 1 *Our offline sequential change detection algorithm takes $O(u)$ space and $O(1)$ per-item processing time to detect local changes, where $u = |U|$.*

Streaming Algorithm. The high-level idea of the streaming algorithm for local changes is similar to that for global change detection. Therefore, data structures and the update procedure we use here are the same as those used in Section 5.3.3. And we shall make changes to the detection procedure. Again we assume that at time n, we have m buckets whose timestamps are after w, $B_1 \ldots B_m$, ordered from the most- to the least-recent; each $B_i = (s_i, t_{m-i+1}, \mathrm{CM}_i)$. The local change detection algorithm consists of two components. For each new item x_n:

1. Estimate $\hat{Z}_{x_n,\lambda}^n$ directly from sketches, for $\lambda = t_1 + 1, \ldots, t_{m-1} + 1$. The intuition comes from the fact that $T_{x_n,\lambda}^n = \sum_{i=\lambda}^n \left(1_{\{x_i = x_n\}} \cdot \log \frac{P_1[x_i]}{P_0[x_i]}\right) = S_\lambda^n[x_n] \cdot \log \frac{P_1[x_n]}{P_0[x_n]}$, where $S_\lambda^n[j] = |\{i | x_i = j, \lambda \leq i \leq n\}|$. We first derive a CM sketch for S_1^n, denoted $\mathrm{CM}_{S_1^n}$, by linearly combining all the sketches in EHCM_{S_1}. Then for each $\lambda = t_q + 1$ ($q = 1..m-1$), we compute CM_λ^n by

the linear combination of sketches: $\mathrm{CM}_\lambda^n = \sum_{j=q+1}^{m} \mathrm{CM}_j$. This gives good estimate for S_λ^n. Finally, $Z_{x_n,\lambda}^n$ is estimated through

$$\begin{aligned}\hat{Z}_{x_n,\lambda}^n &= \mathrm{median}_{i=1}^{\log(\log n/\delta)} count_{\mathrm{CM}_\lambda^n}[i, h_i(x_n)] \cdot \\ &\quad \log\left(\frac{count_{\mathrm{CM}_{S_1^n}}[i, h_i(x_n)] + \gamma}{n + \gamma \cdot u} \cdot count_{\mathrm{CM}_{1/P_0}}[i, h_i(x_n)]\right).\end{aligned}$$

2. Change-point estimate. Having $\hat{Z}_{x_n,\lambda}^n$ for $\lambda = t_1 + 1 \ldots t_{m-1} + 1$, the missing estimates for $\hat{Z}_{x_n,\lambda}^n$ for other λ-values ($w < \lambda \leq n$) are approximated by interpolation with a cubic spline. After interpolation, we can use any known method to find the local maxima. If the local maximum is larger than the threshold τ_l, we report a local change in $P_0[x_n]$, and the maximizer is the change-point estimate.

Corollary 2 *Our streaming change detection algorithm takes* $O(\frac{\log n}{\epsilon} \log \frac{\log n}{\delta})$ *space to detect a local change and to estimate the associated change point, with per-item processing time of* $O(\frac{\log n}{\epsilon} \log \frac{\log n}{\delta})$.

5.5 Global Change Detection Experiments

In this section we evaluate the efficacy of our proposed method SPRT (Section 5.3.2) and its streaming heuristic (Section 5.3.3) which we dub EHCM for detecting global changes against the alternative approaches. The datasets consist of both synthetic data, where we can control change events and see how the change detection algorithms react to different changes, and real data to reveal insights on their applicability and their use.

5.5.1 Experiment Setup

For control purposes, we first evaluate the performance of our methods on synthetic data sets. We can apply SPRT and EHCM to various standard distributions, with finite (Normal, Poisson, Exponential distributions) as well as infinite

variances (Zipf [32], a heavy-tailed distribution). Due to space limit, we demonstrate results from one distribution family—Zipf—in depth, with parameter values ranging from 0 (uniform) to 3 (highly skewed). This distribution is ubiquitous in real applications, such as internet traffic, text streams (e.g. weblogs), citation distributions to web accesses for different sites, etc. [32]. Results on other distributions are similar.

In our experiments, the underlying distribution is a Zipf with parameter value z; the domain size is 10K. Ideally, there is no change in distribution if its z-value keeps constant. However with real applications, it is rare to find a data set whose empirical probability distribution remains unchanged at any time. For robustness, we generalize the concept of no change in distribution: we generate different amount of noise corruptions from a normal distribution $N(0, \sigma^2)$, where $\sigma^2 \in [0, 1]$, to the underlying Zipf z, denoted (z, σ^2). We typically consider two sets of distributions: $A_z = \{(z, i\sigma_0^2)|i = 0, 1, ..., 10,\ \sigma_0^2 = 0.1\}$; $B = \{(i\sigma_0^2, 0)|i = 0, 1, ..., 30,\ \sigma_0^2 = 0.1\}$. Any switch between distributions in A_z, for a given z, is considered *change-free*, whereas a data set is *change-contained* if the distribution changes from one to another within B.

In each of the following experiments, we generate 10 streams with 100K points each; the first and second w =50K points of each stream are generated from $P_0 = (z_0, 0)$ and P_1, respectively. We consider two scenarios: in the i-th change-contained stream, $P_1 = (z_0 + \Delta z, 0)$; in the i-th change-free stream, $P_1 = (z_0, \Delta z),\ \Delta z = 0.1..1$.

5.5.2 Efficacy of Proposed Methods

Accuracy of detected global changes. With 10 change-contained streams, our change detection algorithms (both offline and streaming) never miss detecting a single change. So the miss detection rate on this data set is 0. We repeat the experiment on change-free data sets. Ideally, there should be no alarm triggered,

therefore corresponding number of alarms, after being normalized by w, indicates the false alarm rate of our algorithms. We plot it in Figure 5.4 as a function of Δz, for $z_0 = 1.2$. We also observe very similar results with other z_0 values. Here user-desired miss detection probability β is fixed to 0.05, and we vary user-desired false alarm rate α to decide threshold. Indeed, the observed false alarm rate increases as α gets larger, but is always bounded by α (in the worst case within 0.05%). It is more interesting to observe that with an increased amount of noise, the false alarm rate from change-free data sets flattens out. This reflects the robustness of our methods to noise. Moreover, the approximation error resulting from the underlying sketch technique by comparing the false alarm rate obtained with and without sketch structure is negligible (as illustrated by dashed curves for EHCM in Figure 5.4), in the worst case within an absolute difference of 0.05%.

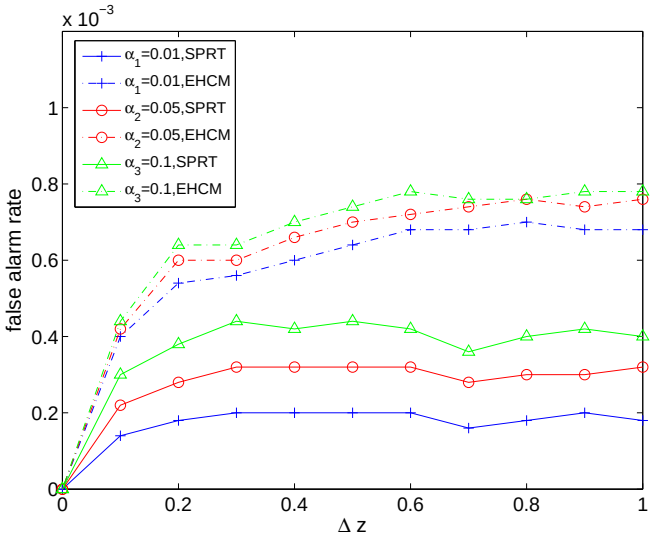


Figure 5.4: Change detection accuracy.

Change point estimates and adaptive windows. For sequential methods, questions about how accurate the change point estimate is and how effective it is to adapt to various rates of change are answered in Figure 5.5. In this experiment, $P_0 = (z_0, 0)$ and $P_1 = (z_0 + \Delta z, 0)$, $\Delta z = 0.1..1$. We report results when $z_0 = 1.2$. Our change detection methods output change point estimate λ^* and detection

time n for each detected change. Knowing that the true change point is at w, the *change point delay* is defined as $\frac{|\lambda^* - w|}{w}$. Likewise, *detection delay* refers to $\frac{n-w}{w}$. It is interesting to observe that all (offline) change-point estimates are close to their true values, independent of individual change rate; whereas faster changes have shorter delays, as we expect. Again, streaming heuristic EHCM behaves equally well with its offline counterpart, across all Δz's.

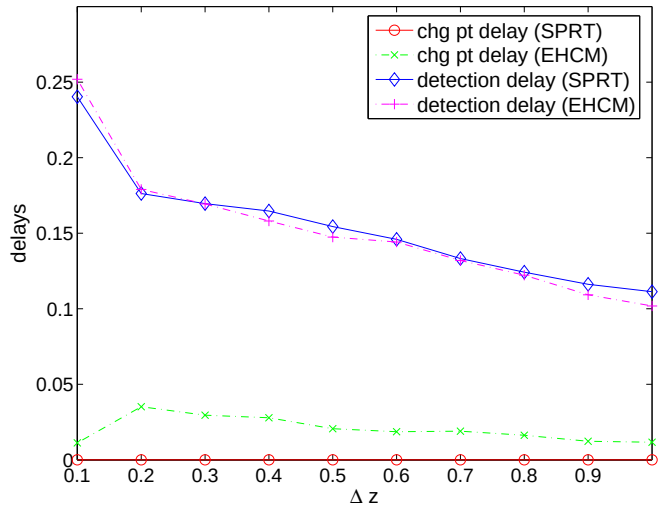


Figure 5.5: Accuracy for global change point estimates and detection delays.

5.5.3 Comparison of Methods

We consider three alternative approaches for change detection, and compare them against our proposed SPRT and EHCM methods both in concept and in practical performance:

- **Fixed-cumulative window model (CWIN).** Given input stream $A_1^n = (x_1, ..., x_n)$, $w < n$, P_0 and P_1 are empirical probability distributions constructed from A_1^w (a fixed window) and A_1^n (a cumulative window). This setup is the same as our SPRT and EHCM. But CWIN computes KL-distance [39] between P_0 and P_1 at every new item x_n, and signal an alarm when the distance is significantly large.

- **Fixed-sliding window model (FSWIN).** This algorithm has a similar flavor to CWIN, but differs in the way that P_1 is built based on the most recent w observations $A^n_{n-w+1} = (x_{n-w+1} \dots x_n)$, which forms a sliding window of size w.

- **Benchmark Sequential Method (BSM).** Different from our algorithms, BSM assumes that not only P_0 but also P_1 is known as *a priori*, so that it guarantees optimum in terms of detection delay [109].

Detection delay. We compare them against our SPRT and EHCM. Again in this experiment, $P_0 = (z_0, 0))$, $P_1 = (z_0 + \Delta z, 0)$, $\Delta z = 0.1..1$, and the true change point is at w. Without knowing the true threshold to trigger significant changes with window-based methods (i.e., CWIN and FSWIN), we adopt a conservative threshold — the KL-distance between distributions $(z_0, 0)$ and $(z_0, 1)$. This is smaller than the true threshold, since we consider it change-free by adding random noise $N(0, 1)$ to the original data. So the detection delay using the conservative threshold is shorter than the true delay. We compare it against detection delays from other methods, and infer the comparative results when using the true threshold.

All methods start detecting changes since x_{w+1}. Figure 5.6 demonstrates the results when $z_0 = 1.2$. As expected, the detection delay of each method decreases with the increase of Δz. CWIN yields the longest detection delay, and even misses detections in the first three streams when change is small. This is because points prior to the change point decay the amount of change in distribution. Therefore FSWIN performs better, since it forgets all stale data that arrive at least w time steps ago. And we can infer that delays from a change detection algorithm with decaying-weighted window will fall between those from FSWIN and CWIN. However, FSWIN's detection delay is still roughly twice of the delay from equally-well-performed SPRT and EHCM. This indicates the effectiveness

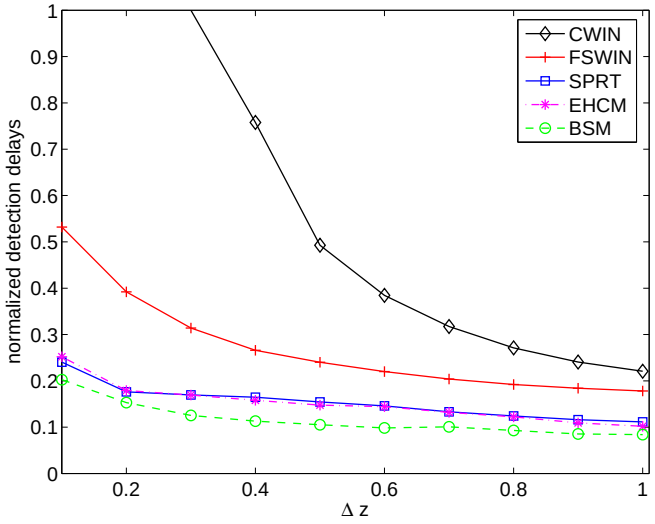


Figure 5.6: Normalized detection delays $\left(\frac{|n-w|}{w}\right)$ of various methods.

of a test statistic when the change-point estimate is integrated, even though the estimate for the associated P_1 may be out-of-date to some degree. Due to the use of a conservative threshold for all window-based solutions, their true gap from sequential methods should get enlarged with the real threshold. There is no surprise that BSM always achieves the minimum delay, but it is interesting to see that detection delays from SPRT/EHCM are mostly close to BSM. This is another strong indicator of prompt response of sequential approaches to changes. The limitation of CWIN method is apparent; hence, we do not consider it in the remainder of our experiments.

5.5.4 Applications

In this section, we study change detection algorithms on real data sets to demonstrate the effectiveness and the applicability of our approaches. In particular, we first measure how well an effective change detection algorithm can quickly and accurately adapt query approximation to the distribution after a change, so as to improve query qualities. This is very crucial for continuous model-based query answering. Then we show the potential power of our methods in applications of

intrusion detection.

Query quality. We obtain the world cup '98 HTTP request logs from the Internet Traffic Archive [76]. We take hourly requests from one server, and focus on the `ClientID` attribute contained in each HTTP request record. Analyzing the data offline finds that logs usually present different distributions in `ClientID` across time. We concatenate two one-hour log files $S_0 = (x_1 \dots x_w)$ and $S_1 = (x_{w+1} \dots x_n)$ for our experiments; $w+1$ is the true change point. We have about 9K distinct `ClientID` in this data set, on which we compare FSWIN and EHCM via a set of queries. Our goal for each query is to quickly and accurately answer it according to the post-change distribution, when it occurs. The ground truth is query results $\mathcal{Q}$ answered from S_1. For each change detection algorithm to be compared, we first run it on the concatenated data set, with P_0 being the empirical distribution constructed from S_0, to get a change-point estimate λ, from which we start to estimate query answers $\hat{\mathcal{Q}}$ based on all observations after λ. In particular, λ output from EHCM is an estimate for the most likely change point, whereas FSWIN with a window of size w returns $\lambda = n - w + 1$ if a change is detected at time n.

Here are the set of queries [96] and the criteria we use to evaluate the accuracy of various query answers (U is a data domain of size u):

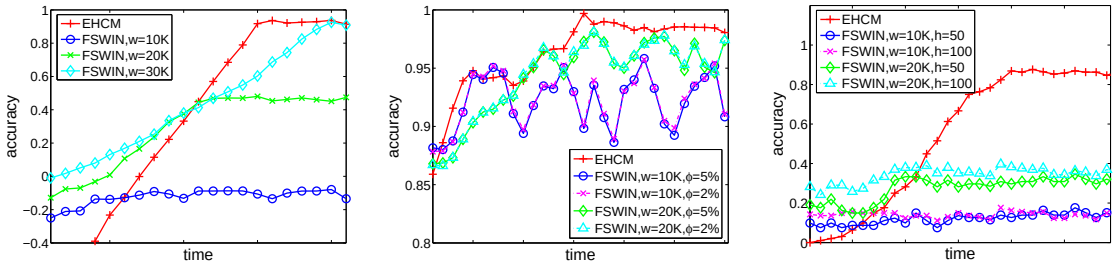
- **Point queries**: let P_2 and $\hat{P}_2$ be empirical probability distributions constructed from respective S_1 and $(x_\lambda \dots x_n)$, where λ is the change-point estimate returned from a change detection algorithm, and $\hat{P}_2$ is an estimate for P_2. A point query $\mathcal{Q}(i)$ is to return an approximation of $P_2[i]$. Thus, the accuracy of point-query estimates is $1 - \frac{1}{u}\sum_{i \in U} \frac{|\hat{\mathcal{Q}}(i) - \mathcal{Q}(i)|}{\mathcal{Q}(i)}$;

- **ϕ-quantiles** ($0 < \phi < 1$): let $F(x) = \sum_{i \in U, i \le x} P_2[i]$. The j-th ϕ-quantile $\mathcal{Q}(j)$ is to find x, such that $F(x-1) < j\phi$, $F(x) \ge j\phi$, $1 \le j \le 1/\phi$. Likewise, $\hat{\mathcal{Q}}(j)$ is an estimate for $\mathcal{Q}(j)$ when P_2 is replaced by $\hat{P}_2$. So the

accuracy of ϕ-quantile estimates is $1 - \phi \cdot \sum_{j=1}^{1/\phi} \frac{|\hat{\mathcal{Q}}(j) - \mathcal{Q}(j)|}{u}$;

- h **heaviest hitters**: a heavy-hitter query returns a set $\mathcal{Q}$ of $i \in U$ that has the h largest $P_2[i]$ values; its estimate based on $\hat{P}_2$ is denoted $\hat{\mathcal{Q}}$. The accuracy for estimating h heaviest hitters is measured by Jaccard coefficient $\frac{|\hat{\mathcal{Q}} \cap \mathcal{Q}|}{|\hat{\mathcal{Q}} \cup \mathcal{Q}|}$ for set similarity (the closer to 1, the better).

We believe that above parameters ϕ, h, and window size w are critical for the evaluation. We analyze the performance under the influence of different parameter values.



(a) point queries (b) ϕ-quantiles (c) h heaviest hitters

Figure 5.7: Accuracy to answer queries on post-change distributions: EHCM vs. FSWIN

Figure 5.7 summarizes the accuracy of EHCM and FSWIN to answer various queries, as a function of time, since the change-point estimate by respective algorithms. It can be seen that: (1) EHCM performs the worst immediately after the change point, due to the lack of data samples from the post-change distribution to estimate $\hat{P}_2$. Shortly after a change occurs, although most items inside the window of FSWIN are outdated, $\hat{P}_2$ from FSWIN may still be better than that from EHCM based on very few samples after λ. It is clear that EHCM gradually outperforms FSWIN and achieves accurate answers to all three queries as more samples are observed. This is because EHCM never underuses points after the change, like what FSWIN does. (2) The performance of FSWIN is dominated by window size w, rather than other parameters, such as ϕ (Figure 5.7(b)) and

h (Figure 5.7(c)): smaller windows (relative to the domain size) lead to inaccurate approximation results, whereas too large windows delay the convergence to accurate answers (see $w = 30K$ in Figure 5.7(a)), since it takes longer for larger windows to "forget" all stale data. All these make EHCM the method of choice. This suggests the importance of accurate change-point estimate in model-based declarative query answering.

5.6 Local Change Detection Experiments

We evaluate in this section the efficacy of our proposed method SPRT and its streaming heuristic that we dub EHCM (Section 5.4) for detecting local changes. We set up experiments with synthetic data as in Section 5.5.1.

Accuracy of Detected Local Changes. We verify the effectiveness of our methods to recognize significant local changes based on prior knowledge on distributions. To avoid enumerating all possible changes, we consider two representative types of changes: type I, small-to-medium changes, in particular from $P_0 = (1.2, 0)$ to $P_1 = (1.3, 0)$; type II, medium-to-large changes, in particular from $P_0 = (1.2, 0)$ to $P_1 = (1.7, 0)$. In both cases, the ground truth is that the set G_k contains singleton items $j \in U$ where top-k local Kullback-Leibler (KL) distance values $d_{KL}(P_1[j]||P_0[j]) = P_1[j] \log(P_1[j]/P_0[j])$ achieve. Figure 5.8(a) shows the top-100 local KL distance values. With this ground truth, we shall evaluate the accuracy of local changes triggered by our local change detection algorithms. Conceptually, all local changes in j should be signaled when its local distance is greater than some threshold. However, this empirical threshold for local KL distance values is not pre-computable, so we propose the evaluation procedure as follows.

Zipf distributions have a property that relatively large local changes happen at the head of a distribution, when switching from P_0 to P_1. Therefore, a given

k-value defines a *change-active domain* $U_k \subset U$ as $U_k = \{0, 1, ..., \max(G_k)\}$, for Zipf distributions, [1] where $\max(G_k)$ is the maximum $j \in G_k$. Furthermore, let H be the set of alarms returned by our method; $H_k = \{j | j \leq \max(G_k), j \in H\} = H \cap U_k$ be a subset of our returned alarms no larger than $\max(G_k)$. Based on these notations, we define evaluation criteria *precision* and *recall* as a function of k (note this is different from traditional definitions):

$$precison_k = \frac{|H_k \cap G_k|}{|H_k|},$$
$$recall_k = \frac{|H_k \cap G_k|}{|G_k|}.$$

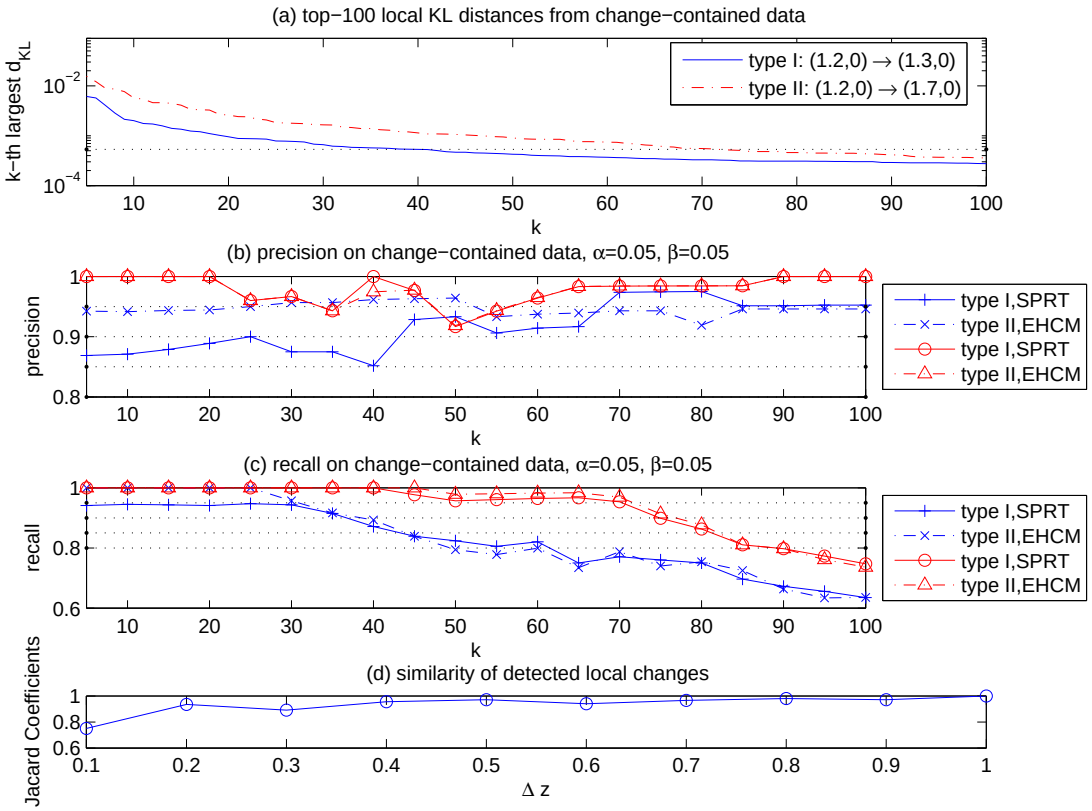


Figure 5.8: Accuracy of detected local changes.

For the two types of changes, Figure 5.8(b)(c) demonstrate the performance of our local change detection algorithms in terms of respective precision and recall,

[1] Change-active domain also exists in other distributions.

when false alarm rate $\alpha = 0.05$, miss detection rate $\beta = 0.05$. It is clear to see that precision and recall from detecting medium-to-large (type II) changes are both higher than those from detecting small-to-medium (type I) changes. This is consistent with our intuition that it is easier to detect larger changes. When focusing on either type of changes, precision increases with k increasing, with the worse case value of 85%, and converges to higher than 95% at large k (as $\alpha = 0.05$ indicates). This suggests H_k become stable (i.e., no more alarms output from our algorithm), while G_k keeps expanding. In contrast, recall starts high (very close to 1) and collapses after some k (i.e., $k = 35$ in type I, $k = 70$ in type II). This indicates that local changes out of these top-k are no longer significant to signal alarms, although they have relatively high ranking in Figure 5.8(a). All these constitute a potential indicator of our accurate and robust change detection algorithms. Moreover, it is interesting to observe that the type I and type II curves in Figure 5.8(a) have very close y-value at $k = 35$ and $k = 70$, respectively, as shown by the horizontal line, which suggests an empirical threshold to significant local changes, under KL distance. We next factor in the additional error resulting from our streaming heuristic EHCM. Figure 5.8(b)(c) show that our streaming algorithm for local change detection performs equally well with its offline counterpart.

Furthermore, we apply both offline and streaming algorithms for detecting local changes to 10 change-contained streams; each has 100K points, the first and the second $w = 50$K points of which are generated from $P_0 = (1.2, 0)$, $P_1 = (1.2 + \Delta z, 0)$, respectively, $\Delta z = 0.1..1$, in increments of 0.1. We compute, for each stream, the similarity between the set of detected local changes by respective SPRT and EHCM. The measurement we use is the *Jaccard coefficient* [41]. High Jaccard-coefficient values, as shown in Figure 5.8(d), indicate the accuracy of our streaming algorithm.

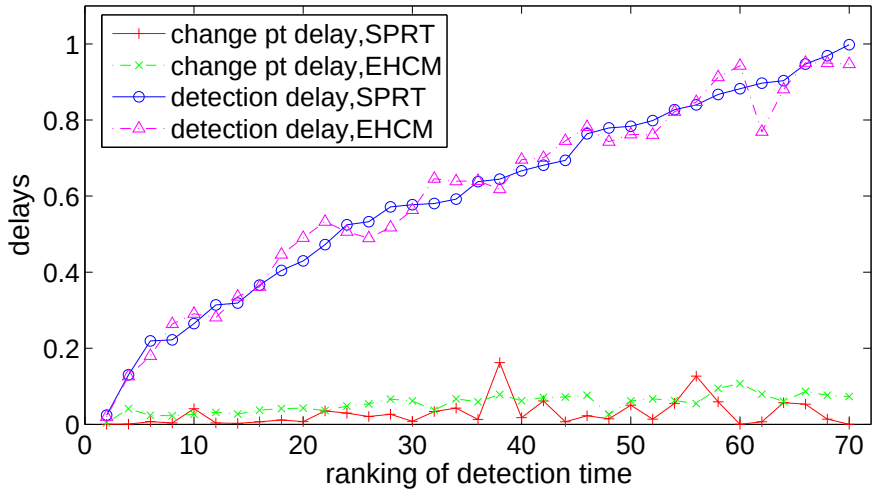


Figure 5.9: Accuracy for local change-point estimates and detection delays.

Change-Point Estimates and Adaptive Windows. With local change detection algorithms, we report in Figure 5.9 their accuracy of change-point estimate and their effectiveness to be adapt to various rates of changes, when $P_0 = (1.2, 0)$, $P_1 = (1.7, 0)$, and omit similar results from other P_1 distributions. Again, we set $\alpha = \beta = 0.05$, and the true change point is at $w = 50K$. Recall that for a local change with change-point estimate λ^* and detection time n, the change point delay is $\frac{|\lambda^* - w|}{w}$; its detection delay is $\frac{n-w}{w}$. Change point delays and detection delays for local changes detected by both SPRT and EHCM are illustrated in Figure 5.9, in order of their detection time with SPRT: a change that is detected early indicates its fast change rate. It is interesting to observe that almost all change point estimates are close to their true values, independent of individual change rate; whereas faster changes have shorter detection delays, as we expect.

Intrusion Detection Application. We repeat experiments on IP traffic data obtained from DARPA intrusion detection evaluation [86]. There are several weeks' worth of IP packet header data. We choose data from two weeks. The first week data is attack free and hence is used to build "base" distribution P_0. The second week data is labelled with the date, starting time, the attack name

and the target destination IP addresses of each attack. [2] We pair up two data sets from the same day of each week, and start our EHCM algorithm to check for any deviation of the second week data from the first.

We focus on two types of attacks that occurred during the day: 'setan' — a network probing attack which looks for well-known weaknesses; and 'neptune' — SYN flood denial of service on one or more ports. Both attacks lead to an increase in packet number in network traffic, so that the distribution of overall packet traffic shifts towards these IP addresses during the attacks. From labelled header data, the ground truth is:

Attack names	**Target IP**	**Starting time**
setan	172.16.114.50	9:33 AM
neptune	172.16.114.207	11:04 AM

When packet count distribution is considered, EHCM detects above attacks at 9:33:32 AM and 11:05:44 AM, with change point estimates being 9:33:32AM and 11:05:27 AM, respectively. This suggests that our proposed method has the potential for intrusion detection monitoring. However, if we apply EHCM to traffic volume (i.e., in terms of number of bytes) distribution, the attack 'setan' is not detected. This indicates that different signals demonstrate different behavior with respect to the changing distribution. Better understanding of how attacks entail changes in underlying distributions is needed to apply sequential change detection to intrusion detection applications.

5.7 Related Work

There is a lot of work on change detection in data streams. Most [54, 80, 39, 31, 84] are based on fixed-size windows: they affix two windows to streaming data and estimate the change in distribution between them. A delay in change detection

[2]Note the difference between changes and attacks. An attack may result in (global or local) changes in distributions; whereas changes may not result from attacks.

is inevitable when the time-scale of change is unavailable. Authors in [54, 80, 39] exploited parallel windows of different sizes to adapt to different change scales. But this is not suitable in data stream context, since we can not afford to explore all window sizes. On the other hand, it has been proposed to use windows of varied sizes [53, 111, 13]. This line of research has to guess *a priori* the time-scale of change, and is mostly computationally expensive. Ho [71] recently proposed a martingale framework for change detection on data streams, where a "pseudo-adaptive" window – delimited by two adjacent change detection points – was used. However, no change point estimate was considered in this work. In contrast to the bulk of this literature, our approaches differ in two key ways. First, when a change is detected, our methods also estimate the most likely change point in time. Second, the change-point estimate is inherent to the evaluation of our test statistic to detect a change, hence, our methods achieve shorter detection delays.

Applying sequential hypothesis testing to change detection is a natural approach. Authors in [79] studied sequential change detection method offline, under a simpler scenario: (1) both pre- and post-change distributions are known as *a priori*; (2) the first observation is assumed to be the change point. Our work here is more general, being able to estimate the actual change point anywhere which is the crux of the problem. In addition, our challenge is to adapt the well-known technique to the streaming scenario, even without independence assumption. Another relevant work is [19], where they adopted exponential histograms to detect change of various scales. However, their approach does not work for data with large domain size, which is our crucial focus.

For local change detection, [31, 84] proposed window-based solutions to find relatively large local changes. However, they triggered a large local change at a singleton item when its difference is at least a user-specified fraction of the sum of differences of all items. This may not indicate a change in the underlying distribution, since the total difference may be small. In contrast, we extend

sequential probability ratio test to both offline and streaming algorithms to detect statistically significant local changes.

5.8 Chapter Summary

There has been plenty of work on techniques for detecting changes in data, but most, if not all, focus on certain "windows" to define and study changes in distributions. This is limiting since window size is a parameter that needs to be determined which is nontrivial, and its size is a lower bound on the detection delay. We have adopted the sound statistical method of sequential hypothesis testing to yield fast and space-efficient algorithms on streams, to detect both global and local changes, without explicit window specification. Our methods are oblivious to data's underlying distributions, and work without independence assumption. Our detailed study of these methods in practice with synthetic and real data sets reveals their effectiveness in detecting changes and many insights on their applicability and their use, depending on the nature of change in the distribution these phenomena induce.

Chapter 6
Conclusions and Future Work

Nowadays, many organizations have applications that deal with millions or even billions of transactions every day. Under this scenario, a lot of work on designing techniques for processing and mining such streaming data has emerged in the past decade. Prior work in streaming analysis has shown how to estimate simple statistics such as frequent items, quantiles, histograms, join sizes etc. on data streams, but very little has focused on more sophisticated statistical analyses that are performed in near real-time, using limited resources. We present streaming techniques for such statistical modeling in this dissertation.

In our study, we investigate two representative categories of modeling in data streams: parametric and structural modeling.

For parametric modeling, there are hierarchical as well as non-hierarchical models. We present highly efficient, small-space approaches for estimating respective model parameters. To the best of our knowledge, this is the first work on model fitting over data streams with *a priori* error guarantees. In addition, we propose online method for model validation so that models can be validated frequently.

For structural modeling, we focus on communication graphs. With the abundance of communication between individuals, many applications on analyzing the patterns behind the communications emerge. The idea of using signatures to capture behavior is an attractive one. Instead of proposing a signature scheme for a particular task, we take a systematic way to study the principles behind the

usage of signatures to any task. In particular, we define fundamental properties of signatures and study a broad set of applications based on what properties they need for signatures to be useful. We explore several signature schemes in our framework and evaluate them on real data in terms of these properties. This provides insights into suitable signature schemes for desired applications. Our experiments demonstrate that no single signature scheme fits all applications; different signatures are needed, depending on the balance among the properties they provide.

Another critical issue in modeling data streams is to detect its change. There has been plenty of work on detecting changes in streams, but most, if not all, focus on certain "windows" to detect the change in distribution between them. This is problematic since window size is a parameter that needs to be determined which is nontrivial. We therefore adopt statistically sound sequential probability ratio test to yield fast and space-efficient streaming algorithm to detect both global and local changes, without explicit window specification. Our methods are oblivious to data's underlying distributions, and work without the independence assumption. Another advantage of our proposed sequential approaches is shorter detection delays. This is due to the integration of most likely change-point estimate into the test statistics of our solutions.

As concrete applications of our techniques, we focus on network traffic data, one of the richest sources for data streams. We complement our analytic and algorithmic results with experiments on them to demonstrate the practicality of our methods at line speeds, and potential power of streaming techniques for statistical modeling in data mining.

In future work, we propose to study the following.

- Parametric modeling: widely used general modeling procedures that involve recursive computing such as expectation maximization, as well as regression models other than linear regression presented in this dissertation, will be

explored.

- Structural modeling: we will consider the temporal dimension on signatures in communication graphs. Interesting questions include but are not limited to "What is the signature at a given time interval?", "What sequence of activities capture my communication behavior?", etc. Another line of research is to study hierarchies of signatures over time. Furthermore, advanced hashing is a promising technique to speed up nearest-neighbor searches among signatures.

- Change detection: Almost all real world data sets are beyond single dimension, so we shall address the problems with higher dimensions in our future work.

Bibliography

[1] STREAM: Stanford stream data manager, http://www-db.stanford.edu/stream/.

[2] Sprint CMON, http://www.sprintlabs.com/.

[3] http://www.streambase.com/.

[4] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, and S. Zdonik. Aurora: a data stream management system. In *SIGMOD*, 2003.

[5] L. Adamic. Zipf, power-law, pareto - a ranking tutorial. `http://www.hpl.hp.com/research/idl/papers/ranking/`, 2000.

[6] N. Alon, N. Duffield, C. Lund, and M. Thorup. Estimating sums of arbitrary selections with few probes. In *ACM PODS*, pages 317–325, 2005.

[7] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *JCSS*, 58:137–147, 1999.

[8] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *PODS*, 2004.

[9] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, 2002.

[10] Ziv Bar-Yossef. Sampling lower bounds via information theory. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 335–344, 2003.

[11] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *RANDOM '02: Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, pages 1–10, London, UK, 2002. Springer-Verlag.

[12] P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in web client access patterns characteristics and caching implications. In *WWW*, 1999.

[13] P.L. Bartlett, S. Ben-David, and S.R. Kulkarni. Learning changing concepts by exploiting the structure of change. In *Computational Learing Theory*, pages 131–139, 1996.

[14] M. Basseville and I.V. Nikiforov. *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, Inc.

[15] J. Baumes, M. Goldberg, M. Hayvanovych, M. Magdon-Ismail, W.A. Wallace, and M. Javeed Zaki. Finding hidden group structure in a stream of communications. In *ISI*, 2006.

[16] A. Belussi and C. Faloutsos. Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In *VLDB*, 1995.

[17] P. Berkhin. A survey on pagerank computing. *Internet Mathematics*, 2(1):73–120, 2005.

[18] Z. Bi, C. Faloutsos, and F. Korn. The "dgx" distribution for mining massive, skewed data. In *KDD*, 2001.

[19] A. Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *SDM*, 2007.

[20] A. Chakrabarti, K. Do Ba, and S. Muthukrishnan. Estimating entropy and entropy norm on data streams. In *STACS*, pages 196–205, 2006.

[21] A. Chakrabarti, G. Cormode, and A. McGregor. A near-optimal algorithm for computing the entropy of a stream. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, pages 328–335, 2007.

[22] Kaushik Chakrabarti, Minos Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate query processing using wavelets. *The VLDB Journal*, 10(2-3):199–223, 2001.

[23] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S.R. Madden, F. Reiss, and M.A. Shah. TelegraphCQ: continuous dataflow processing. In *SIGMOD*, 2003.

[24] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *ICALP*, 2002.

[25] C.-M. Chen, H. Agrawal, M. Cochinwala, and D. Rosenbluth. Stream query processing for healthcare bio-sensor applications. In *ICDE*, 2004.

[26] CIDR, http://www.webopedia.com/TERM/C/CIDR.html.

[27] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *PODS*, 2003.

[28] D. Coppersmith and R. Kumar. An improved data stream algorithm for frequency moments. In *SODA*, 2004.

[29] G. Cormode, T. Johnson, F. Korn, S. Muthukrishnan, O. Spatscheck, and D. Srivastava. Holistic udafs at streaming speeds. In *SIGMOD*, 2004.

[30] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *LATIN*, 2004.

[31] G. Cormode and S. Muthukrishnan. What is new: Finding significant differences in network data streams. In *INFOCOM*, pages 1534–1545, 2004.

[32] G. Cormode and S. Muthukrishnan. Summarizing and mining skewed data streams. In *SDM*, 2005.

[33] G. Cormode, S. Muthukrishnan, and I. Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *VLDB*, 2005.

[34] Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). *IEEE Transactions on Knowledge and Data Engineering*, 15(3):529–540, 2003.

[35] C. Cortes and D. Pregibon. Signature-based methods for data streams. *Data Min. Knowl. Discov.*, 5(3):167–182, 2001.

[36] Corinna Cortes, Daryl Pregibon, and Chris Volinsky. Communities of interest. *Lecture Notes in Computer Science*, 2189, 2001.

[37] C. D. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *SIGMOD*, 2003.

[38] Chuck Cranor, Yuan Gao, Theodore Johnson, Vlaidslav Shkapenyuk, and Oliver Spatscheck. Gigascope: high performance network monitoring with an sql interface. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 623–623, 2002.

[39] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi. An information-theoretic approach to detecting changes in multi-dimensional data streams. In *Interface*, 2006.

[40] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *SODA*, 2002.

[41] M. Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. In *ESA*, 2002.

[42] A. Deshpande, C. Guestrin, and S. Madden. Using probabilistic models for data management in acquisitional environments. In *CIDR*, 2005.

[43] A. Deshpande, C. Guestrin, S. Madden, J.M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[44] A. Deshpande, C. Guestrin, S. Madden, J.M. Hellerstein, and W. Hong. Model-based approximate querying in sensor networks. *VLDB Journal*, 2005.

[45] M. Evans, N. Hastings, and B. Peacock. *Statistical Distributions*. Wiley, New York, 3rd edition, 2000.

[46] C. Faloutsos and I. Kamel. Beyond uniformity and independence: Analysis of R- trees using the concept of fractal dimension. In *PODS*, 1994.

[47] C. Faloutsos, Y. Matias, and A. Silberschatz. Modeling skewed distributions using multifractals and the '80-20 law'. In *VLDB*, 1996.

[48] T. Fawcett and F.J. Provost. Activity monitoring: Noticing interesting changes in behavior. In *SIGKDD*, 1999.

[49] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the streaming model: the value of space. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 745–754, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[50] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, 2005.

[51] P. Flajolet and G.N. Martin. Probabilistic counting. In *FOCS*, 1983.

[52] X. Gabaix, P. Gopikrishnan, V. Plerou, and H.E. Stanley. A theory of power law distributions in financial market fluctuations. 423:267–270, 2003.

[53] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *Lecture Notes in Computer Science*, 2004.

[54] V. Ganti, J.E. Gehrke, R. Ramakrishnan, and W.-Y. Loh. A framework for measuring changes in data characteristics. *Journal of Computer and System Sciences*, 64:542–578, 2002.

[55] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *SIGMOD*, 2001.

[56] P.B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB*, 2001.

[57] P.B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on External Memory Algorithms and Visualization*, volume A, 1999.

[58] Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 281–291, 2001.

[59] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, 2005.

[60] A.C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *STOC*, pages 389–398, 2002.

[61] A.C. Gilbert, W. Willinger, and A. Feldmann. Scaling analysis of conservative cascades, with applications to network traffic. In *IEEE Transaction on Information Theory*, volume 45, pages 971–991, 1999.

[62] T. Krishnan G.J. McLachlan. *The EM Algorithm and Extensions*. Wiley-Interscience, 1996.

[63] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD*, 2001.

[64] Sudipto Guha. Space efficiency in synopsis construction algorithms. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 409–420, 2005.

[65] Sudipto Guha, Nick Koudas, and Kyuseok Shim. Data-streams and histograms. In *ACM Symposium on Theory of Computing*, pages 471–475, 2001.

[66] Sudipto Guha, Andrew McGregor, and Suresh Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 733–742, 2006.

[67] Taher H. Haveliwala. Topic-sensitive pagerank. In *WWW*, 2002.

[68] S. Hill, D. Agarwal, R. Bell, and C. Volinsky. Building an effective representation for dynamic network. *Computational and Graphical Statistics*, 15(3):584–608(25), 2006.

[69] S. Hill and F. Provost. The myth of the double-blind review? Author identification using only citations. *SIGKDD Explorations*, 5(2):179–184, 2003.

[70] S. Hill, F. Provost, and C. Volinsky. Network-based marketing: Identifying likely adopters via consumer networks. *Statistical Science*, 21(2):256–276, 2006.

[71] S. Ho. A martingale framework for concept change detection in time-varying data streams. In *ICML*, pages 321–327, 2005.

[72] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *SIGKDD*, 2001.

[73] A. Hussain, J. Heidemann, and C. Papadopoulos. Identification of repeated dos attacks. In *INFOCOM*, 2006.

[74] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *FOCS*, pages 189–197, 2000.

[75] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, 1998.

[76] Internet traffic archive. `http://ita.ee.lbl.gov`.

[77] A.K. Jain, R.W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[78] C. Jr, A. Traina, L. Wu, and C. Faloutsos. Fast feature selection using the fractal dimension. In *SBBD*, 2000.

[79] J. Jung, V. Paxson, A.W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE symposium on security and privacy*, 2004.

[80] D. Kifer, S. Ben-David, and J. Gehrke. Detecting changes in data streams. In *VLDB*, 2004.

[81] E. Kohler, J. Li, V. Paxson, and S. Shenker. Observed structure of addresses in ip traffic. In *IMC*, 2002.

[82] F. Korn, S. Muthukrishnan, and Y. Wu. Modeling skew in data streams. In *SIGMOD*, 2006.

[83] B. Krishnamurthy, H. Madhyastha, and O. Spatscheck. ATMEN: a triggered network measurement infrastructure. In *WWW*, 2005.

[84] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: methods, evaluation, and applications. In *IMC*, 2003.

[85] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *SIGCOMM*, 2004.

[86] DARPA intrusion detection evaluation. `http://www.ll.mit.edu/IST/ideval/index.html`.

[87] F. Li, C. Chang, G. Kollios, and A. Bestavros. Characterizing and exploiting reference locality in data stream applications. In *ICDE*, 2006.

[88] T. Liu, C.M. Sadler, P. Zhang, and M. Martonosi. Implementing software on resource-constrained mobile sensors: Experiences with impala and zebranet. In *MobiSys*, 2004.

[89] D. Madigan. DIMACS working group on monitoring message streams. `http://http://stat.rutgrs.edu/~madigan/mms/`.

[90] B.B. Mandelbrot. *Fractals and Scaling in Finance*. Springer-Verlag, New York, 1997.

[91] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, 2002.

[92] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 251–262, 1999.

[93] S.J. Mason and N.E. Graham. Areas beneath the relative operating characteristics (ROC) and relative operating levels (ROL) curves: Statistical significance and interpretation. *Q. J. R. Meteorol. Soc*, 30:291–303, 1982.

[94] MassDAL poster. `http://www.cs.rutgers.edu/~muthu/massdalposter.pdf`.

[95] R. Motwani and P. Raqhavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[96] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[97] Cisco netflow. `http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm`.

[98] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Symposium on Security and Privacy*, 2005.

[99] C.C. Noble and D.J. Cook. Graph-based anomaly detection. In *SIGKDD*, 2003.

[100] B.-U. Pagel, F. Korn, and C. Faloutsos. Deflating the dimensionality curse using multiple fractal dimensions. In *ICDE*, 2000.

[101] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *ICDE*, 2003.

[102] S.I. Resnick. Heavy tail modeling and teletraffic data. *The Annals of Statistics*, 25:1805–1869, 1997.

[103] M. Roughan and C. Kalmanek. Pragmatic modeling of broadband access traffic. In *Computer Communications 26(8)*, 2003.

[104] M. Schroeder. *Fractals, Chaos, Power Laws: Minutes From an Infinite Paradise*. W.H. Freeman and Company, New York, 1991.

[105] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: new aggregation techniques for sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249, 2004.

[106] D. Song, P. Venable, and A. Perrig. User recognition by keystroke latency pattern analysis. 1997.

[107] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Relevance search and anomaly detection in bipartite graphs. *SIGKDD Explorations Special Issue on Link Mining*, 7(2):48–55, 2005.

[108] Chayant Tantipathananandh, Tanya Berger-Wolf, and David Kempe. A framework for community identification in dynamic social networks. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 717–726, 2007.

[109] A. Wald. *Sequential Analysis*. Dover Publications, 2004.

[110] M. Wang, T. Madhyastha, N.H. Chan, S. Papadimitriou, and C. Faloutsos. Data mining meets performance evaluation: Fast algorithm for modeling bursty traffic. In *ICDE*, 2002.

[111] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[112] W. Willinger, D. Alderson, and L. Li. A pragmatic approach to dealing with high-variability in network measurements. In *IMC*, 2004.

[113] W. Willinger and V. Paxson. Where mathematics meets the internet. *Notices of the American Mathematical Society*, 45(8):961–970, 1998.

[114] W. Willinger, V. Paxson, and M.S. Taqqu. *Self-similarity and Heavy Tails: Structural Modeling of Network Traffic*. Chapman & Hall, New York, 1998.

[115] A. Wong, L. Wu, P.B. Gibbons, and C. Faloutsos. Fast estimation of fractal dimension and correlation integral on stream data. In *Inf. Process. Lett. 93(2): 91-97*, 2005.

[116] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 407–418, 2006.

[117] K. Xu, Z. Zhang, and S. Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. *SIGCOMM Comput. Commun. Rev.*, 35(4):169–180, 2005.

[118] Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR*, 2003.

[119] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369, 2002.

Curriculum Vita

Yihua Wu

EDUCATION

October 2007 Ph.D. in Computer Science, Rutgers University, U.S.A.

May 2003 M.S. in Computer Science, Rutgers University, U.S.A.

July 2000 B.S. in Computer Science, Peking University, Beijing, P.R. China

EXPERIENCE

Jan.2007—Aug.2007 Research Assistant, Department of Computer Science, Rutgers University, New Brunswick, NJ

Oct.2004—Sep.2007 Research Consultant, Database Management Research Department, AT&T Shannon Research Lab, Florham Park, NJ

Jan.2006—Dec.2006 Teaching Assistant, Department of Computer Science, Rutgers University, New Brunswick, NJ

Jan.2005—Dec.2005 Research Assistant, Department of Computer Science, Rutgers University, New Brunswick, NJ

Sep.2001—Dec.2004 Teaching Assistant, Department of Computer Science, Rutgers University, New Brunswick, NJ

Jan.2001—Apr.2001 Software Engineer, Asia Electronic Commerce Ltd., Beijing, P.R. China

PUBLICATION

On Signatures for Communication Graphs. Graham Cormode, Flip Korn, S. Muthukrishnan and Yihua Wu. Submitted.

Sequential Change Detection on Data Streams. S. Muthukrishnan, Eric van den Berg and Yihua Wu. In International Conference on Data Mining (ICDM) Workshop on Data Stream Mining and Management, 2007.

Modeling Skew in Data Streams. Flip Korn, S. Muthukrishnan and Yihua Wu. In Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD), 2006.

Fractal Modeling of IP Network Traffic at Streaming Speeds. Flip Korn, S. Muthukrishnan and Yihua Wu. In Proceedings of the 22nd IEEE Conference on Data Engineering (ICDE), 2006.