DIMACS Technical Report 2001-43
November 2001

QuickSAND: Quick Summary and Analysis of Network Data

by

A. C. Gilbert, Y. Kotidis, S. Muthukrishnan[1], M. J. Strauss[2]
AT&T Labs-Research
{agilbert, kotidis, muthu, mstrauss}@research.att.com

[1]Permanent Member
[2]Permanent Member

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, NEC Research Institute and Telcordia Technologies (formerly Bellcore).

DIMACS was founded as an NSF Science and Technology Center, and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

Monitoring and analyzing traffic data generated from large ISP networks imposes challenges both at the data gathering phase as well as the data analysis itself. Still both tasks are crucial for responding to day to day challenges of engineering large networks with thousands of customers. In this paper we build on the premise that approximation is a necessary evil of handling massive datasets such as network data. We propose building compact summaries of the traffic data called *sketches* at distributed network elements and centers. These sketches are able to respond well to queries that seek features that stand out of the data. We call such features "heavy hitters." In this paper, we describe sketches and show how to use sketches to answer aggregate and trend-related queries and identify heavy hitters. This may be used for exploratory data analysis of network operations interest. We support our proposal by experimentally studying AT&T WorldNet data and performing a feasibility study on the Cisco NetFlow data collected at several routers.

1 Introduction

The Internet is a vast decentralized, self-configuring, stateless and connectionless entity with which most of us interact. It is loosely structured into autonomous systems and the Internet service providers form clouds in this vast space. In order to have operational control over their network, the service providers have to understand the relationship between their network and others around them and understand the dynamics, faults and traffic patterns within and across their networks. This is a daunting task for most large service providers. The networking research community has just begun to make progress on building models for traffic [10], identifying the principles that govern network dynamics [7], developing predictive mechanisms, etc. In the absence of well developed guidelines for understanding network behavior and sophisticated automatic tools for network management, service providers have to rely on monitoring their network and analyzing the traffic data generated by their network to respond to day to day (even minute to minute!) challenges of engineering a large network with customers.

Network data based engineering and operations of IP networks face two challenges.

1. *Data Gathering.* This is an arduous task. First, the data are massive: the collection of packet and flow traces, and routing and configuration tables, which change rapidly, can add up to terabytes per week. Second, the network infrastructure to collect data is sparse since it is expensive to monitor all parts of the network. Finally, the data are unreliable because the transport mechanism, typically UDP, that is employed to move the data to a central place is not dependable – packets get lost, delayed or delivered out of order, measurement points are not always reachable, routes go down, etc.

2. *Data Analysis.* This is a complex task because many data sources have to be merged and collated to piece together hypotheses on network behavior. Large amounts of data may swamp significant signals. More data does not necessarily mean more information. Forming hypotheses is hard since we do not have a principled approach to IP network engineering. In addition, we do not have good database models in which to pose queries. Finally, the massive amount of data makes run time and performance critical issues. Analysis methods must scale.

Despite these difficulties, there are increased ways to probe the network, set more alarms, and collect more statistics. Large service providers are strengthening their data gathering and analysis infrastructures. This promises to be a long process and the research community must deliver more principles for network data based engineering. While adding tremendous measurement capability, network operators routinely face very basic questions about the network: What are the large flows at a router? What is the difference between traffic at different routers, different times? If we add a customer how will the loads change? Where are the hot spots? Are we adhering to various peer agreements this week?

It is clear that ISPs have fundamental questions about data-based network engineering and operation and that the research community is not yet capable of answering these questions. Our premise is that we can ask these questions differently and still return useful answers. In this paper, we will describe a novel proposal for distributed data gathering as well as analysis. As a result of the challenges presented by data gathering, the community of researchers as well as network operators have grown accustomed to the concept of approximation, whether through sampling, filtering, or aggregating network data. Could this uncertainty be used in a different way? We hypothesize that approximation is a necessary evil of massive scale data analysis and gathering. Our proposal involves computing a "sketch" that summarizes the data as it streams by. Sketches

- are of small space (say 100K bytes),
- can be computed very quickly as the data stream by,
- can be collected at distributed centers, and
- can interact in a flexible way with the underlying network infrastructure for network measurement (e.g., can be transported to a central warehouse periodically or queried actively).

With this infrastructure, network managers and operators have instant hands-on access for performing data analysis. In particular, we present a framework to ask any number of queries on the sketches, including

- simple aggregate queries, such as point or range queries ("slice and dice" the data),
- historic, trend-related queries (behavior over time),
- distributed queries (behavior across different routers), and
- clustering or similarity measures of the data.

The sketches are able to respond well to the queries that have a clear signal; i.e., ones that stand out of the data (this will be formalized in terms of signal energy later). We call such features "heavy hitters." With sketches, we cannot, however, find a needle in a haystack but we can provide a good enough answer if that answer is significant in the data. We also have parameters attached to these sketches that allow us to tune the goodness and the significance of the answer. These types of queries provide valuable perspectives of network data. They can be used to generate wavelet coefficients, which themselves can reveal interesting features of network traffic [9, 7, 8]. These queries can also be used to determine the amount of non-TCP friendly traffic traversing a network or whether a given ISP is conforming to peering agreements, for example.

In this paper, we will describe methods for computing and manipulating sketches. We will also describe how to use sketches to answer queries of network interest, keeping in mind that these answers are "heavy hitters" and are approximate. We will support our proposal by looking at AT&T WorldNet data and doing a preliminary feasibility study on the Cisco NetFlow data collected at several routers. We believe that the proposal of using sketches to do network data analysis is an exciting new direction, to be explored further.

Map of the paper. In Sections 2 we give an overview of the infrastructure and current data collection methods in large ISPs; we also abstract the data stream model for processing network data and review existing approaches for network data gathering and analysis. In Section 3 we describe our proposal. In Section 4, we provide experimental evidence on AT&T WorldNet data supporting our proposal. Finally, in Section 5 we conclude.

2 Background

This section gives a brief overview of large ISP backbone architectures. It outlines existing data collection efforts at distributed points throughout an ISP backbone. We also abstract the data stream model inherent in this process. Finally, we review known approaches to data gathering and analysis.

2.1 ISP architecture

A large ISP backbone network consists of hundreds of gateway and access routers at the edge of the network, backbone routers at its core, and thousands of bidirectional layer-three links. The gateway routers connect to neighboring providers (or "peers") and to public access points via peering links. These routers implement routing policies for peering relationships. Access routers serve modem banks, business customers, web-hosting complexes, etc. via access links. This type of router must filter packets based on customers' addresses, enforce traffic limits, and mark packets for QoS. Backbone links connect the backbone routers between and within major cities to facilitate high-speed switching at the core of the network.

The AT&T backbone network is an example of a large operational ISP network. It has hundreds of routers and over thousands of layer-three links. There are a number of modem pools around the country for dial-up ISP customers and web-hosting complexes for both personal and business customers. Figure 1 depicts the network architecture of a typical large ISP.

2.2 Existing data collection at distributed centers

An ISP can collect a variety of data types at a number of sources in its backbone. We focus on passive measurements and outline the current typical methods of collecting this data.

At the finest and most detailed level of observation, packet monitors passively collect IP packets at a single link (with line speed up to 100Mbs) and record IP, TCP/UDP, or application layer traces. An IP header is 20 bytes long and contains twelve fields, four of which are useful in data analysis: total packet length, transport protocol, and source and destination IP addresses. A TCP header is also 20 bytes long with with such useful fields as source and destination port numbers, sequence number, acknowledgment number, ACK/SYN/FIN/RST flags, and window size. "Packet sniffers" are typically separate PCs that can be placed at different points in the network depending on the goal of the resulting

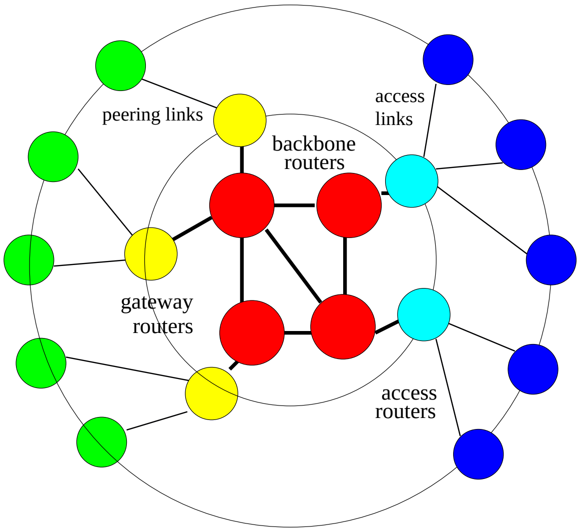

Figure 1: Architecture of a typical large ISP.

data analysis. For example, we can record traffic traces at a modem bank to obtain fine-grain detail about dial-up user behavior, on an access link to monitor an important piece of network architecture, or in front of a web hosting complex to determine where best to place web caches [19]. Because these devices record the header of each packet on a single link, the data are voluminous and difficult to use for an overall picture of network behavior. They do, however, provide useful answers for several important network operational and scientific questions: estimates of throughput and delay of web downloads, sizes of typical web transfers, and burstiness of traffic on the monitored link over time.

At a higher level, we can collect flow data. These data aggregate the packet header information to the IP flow level. A flow is a group of packets from the same port number and source/destination pair wherein each consecutive packet is not separated by more than a certain duration. The packet header statistics we can collect for each flow (using Cisco NetFlow, for example) include source and destination IP addresses, source and destination port numbers, transport protocol, ToS, TCP flags, etc. in addition to the start and end time of each flow and the total number of bytes and packets in the flow. With NetFlow, we are also able to gather routing information for each flow, consisting of the input and output interfaces the flow traversed, source and destination IP prefix, and the source and destination autonomous system numbers. For approximately each 140 bytes of traffic, one byte of flow data is generated; a savings in space compared to the packet monitors. We can gather these data in several different ways: connecting a PC to a router that generates flow records, using line cards that generate these records, or placing a packet monitor between two routers. In the AT&T Backbone we currently have a number of routers in different major cities capable of collecting NetFlow statistics. From the NetFlow data we are able to determine the distribution of applications, as well as the number of bytes (or packets) in each flow, source/destination pairs, load, etc.

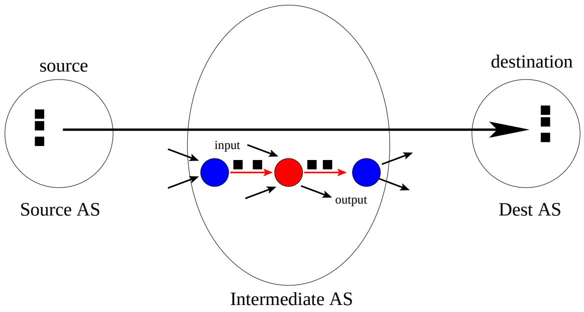

Figure 2: Capturing traffic information in a streaming fashion.

In Section 4 we focus on distributed data collection and analysis of the NetFlow records in AT&T's IP backbone. For this reason we concentrate on the description of packet and flow measurement. All our discussions and methods equally apply to other network data, eg., fault alarms which play great role in the day-to-day management of an IP network.

There are, however, two other types of passive measurements we can instrument. One is routing information, including the routing tables and live feeds of link state advertisements for OSPF and BGP updates. The routing tables and their live feeds are critical for traffic engineering tasks such as load balancing on links to customers and peers. It is difficult to gain an overall view of network dynamics without this information. The second is fault alarms which play an even greater role in the day-to-day management of an IP network.

Finally, coupled with all of these types of passive measurements is the underlying network configuration, topology, and customer and peering agreements. While these pieces of information are very different from passive network measurements, they are crucial components in any distributed data collection and analysis system.

2.3 Data streams

Both the packet header and flow collection methods record information as the traffic streams past the packet monitor or router. See Figure 2 for a picture of this process. In both cases, special software (and sometimes hardware) must be built in order to gather this information at very high speed with high accuracy. This type of data input also creates problems from a data summary and analysis perspective as well. A summary method must be able to work with data presented in a number of streaming forms. Prabhakar et al [20] and Gilbert, et al. [13] formally define different data stream models, including the "cash register" format, which is most relevant given the data collection we have described above. In this data input model, packets (or flows) from different IP sources (or to different IP destinations) arrive in an unaggregated fashion and out of order. For example, the packet monitor does not know how many packets from IP address i arrive during a ten minute period until that period is over, only then can it generate an aggregate number of packets for that address. In addition, those packets from address i pass the monitor interspersed among packets from many other IP addresses.

2.4 Different Approaches for Network Data Collection and Analysis

Analysis of network data must meet a number of stringent requirements that make it impossible to rely on a one-size-fits-all architecture. The data centric approach of analyzing network data relies on building large Data Warehouses. In the broadest sense, a data warehouse is a single, integrated informational store that provides stable, point-in-time data for decision support applications [3].

There are good justifications for such an approach:

- On line analytical processing (OLAP) software allows the collected data to be rendered across any of the collected attributes (e.g. source/destination address/port, protocol, etc.) and at any level of aggregation (e.g., 32bit addresses, subnets, autonomous systems) in an interactive way. This is extremely useful during exploratory data analysis where patterns are typically discovered in an ad-hoc manner.

- Given that the database schema is relatively static there are off-the-shelf tools that we can use for mining interesting patterns, report writing, etc.

- The ability to manipulate information from multiple sources in a single analysis platform enables us to make and test more “interesting” hypotheses.

- Archiving information in a centralized store allows historic trend analysis. Historic information is invaluable not only to find out that something went wrong but also to understand why it happened (e.g. were there any topology/configuration changes at the same period? Was the network affected due to a large customer attached to an access router?).

Building such a gigantic information repository of network measurements imposes several practical difficulties:

- There are many types of data from many sources, including both transactional data such as packet headers or flow records and topological configurations. Not only do these data come from varied sources, but they come in an abundance. A large network has hundreds of routers and thousands of links. With such a large network an extensive measurement infrastructure may be impossible at all points, including modem banks and customer access links. We cannot harm the current performance of the network with measurement tools nor with the delivery of measurement data.

- There are a number of data quality problems once we have built such an infrastructure. It is difficult to obtain precise, synchronized timestamps on a variety of measurements. There are delays in receiving so-called real-time data feeds and these feeds are usually transferred with an unreliable protocol, resulting in significant losses. In some cases we can correct for the lost transmittions by joining multiple datasets. For example, NetFlow records are transmitted to the collection server using UDP. Limited bandwidth results in a loss of up to 90% of the UDP packets during heavy load period. Using link-load statistics from SNMP, we can estimate a correction factor for the received records to account for lost data [10].

In general a data centric approach is good for harvesting interesting trends in the measured data and for time-related trend analysis. On the other hand, it is not suitable for real-time analysis. For such applications a more direct approach is to execute queries at the network interface card (NIC) of a network monitoring device. Processing such as filtering, transformation and aggregation of network traffic can then be performed at the spot. Typically these queries are hard-wired into the NIC, however T. Johnson et al. [17] are implementing new architectures that allow the engineer to load a precompiled query into the device.

Executing queries at the NIC imposes some restrictions too:

- Network monitoring devices are typically entry-level machines with small processing and storage capacities. For example in WordNet we use 500Mhz unix workstations with 10GB disks and 14GB tapes [5]. In comparison the WorldNet Data Warehouse runs in a Sun Enterprise 6000 server with 32 processors, 32GB of main memory and several TBs of disk storage. Thus, we can only expect to perform relatively simple queries at the NIC and defer complex aggregations to the central store.
- Even though data is processed and filtered at the NIC, the query result has to be shipped back to the analyst. This potentially limits the type of queries we can execute to simple coarse aggregations of the data. For example, we can not compute the amount of traffic sent for each pair of source-destination IP addresses because the result in aggregate format is comparable to the size of the NetFlow data.
- Even small output queries might require substantial resources (*scratch space*) for storing temporal results. Given the limited recourses at the NIC, only a few active queries might be possible at any given point.

3 Our Proposal

3.1 Definition of Sketches

Network engineers and researchers have three basic methods for windowing the measurement data: filtering, aggregation, and sampling [15]. We propose a fourth method: sketching.

Sketches are based on random projections. That is, the *sketch* of a vector a is the inner product of a with a suitable collection of random vectors. Random vectors are generatd using "seeds" which can be stored in small space; the properties of sketches differ based on the random variables used [2, 16]. The number of random vectors used is also typically

Figure 3: The array_sketch synopsis data structure.

```
sketch_t initialize( length_t N, distortion_t ϵ, threshold_t τ,
                     failure_prob_t δ);
sketch_t update( sketch_t s, index_t i, value_t v);
sketch_t combine( coefficient_t c1, sketch_t s1, coefficient_t c2, sketch_t s2);
value_t norm( sketch_t s);
value_t cos( sketch_t s1, sketch_t s2 );
value_t query( index_t i1, index_t i2, sketch_t s);
```

small. This process is an example of dimensionality reduction [18], used here to summarize the vector a.[1] Some of the details about sketches and their properties are described later.

3.2 Details of the Array_Sketch Data Structure

Our main tool is a synopsis data structure [11] for an array. It maintains a representation of an array of N elements, and supports updating values of the array and making queries about the values. The answers have a small chance of arbitrary failure, and, when successful, are only approximate. But the size of the data structure is much smaller than a traditional data structure for an array.

The particular data structure we call an *array_sketch*. The data structure supports the following operations, `initialize`, `update`, `combine`, `norm`, `cos`, and `query`, as indicated in Figure 3.

We now describe these operations in detail.

- The `initialize()` function returns a sketch of the all-zeros array of length N (see `query`, below, for the semantics of ϵ, τ, and δ).

- If the sketch s represents the array a, then the `update(` s, i, v`)` function returns a sketch of the array a' gotten by adding v to a_i.

- If sketch s_1 represents array a_1 and sketch s_2 represents a_2, then `combine(` c_1, s_1, c_2, s_2`)` returns a sketch of the vector linear combination $c_1a_1 + c_2a_2$.

- If sketch s represents the array a, then, with probability $1 - \delta$, `norm(s)` returns an approximation to $\sum_i a_i^2$, namely, $(1 \pm \epsilon)\sum_i a_i^2$.

- If sketch s_1 represents vector a_1 and sketch s_2 represents a_2, then, with high probability, `cos(` s_1, s_2 `)` returns an approximation to the cosine of the angle between a_1 and a_2 provided it is large; namely, if $\langle a_1, a_2\rangle^2 \geq \tau\|a_1\|^2\|a_2\|^2$, then, with probability $1 - \delta$, `cos(` s_1, s_2`)` returns $(1 \pm \epsilon)\langle a_1, a_2\rangle^2/(\|a_1\|^2\|a_2\|^2)$.

[1] Other similar techniques include sampling, min-wise independent random variables [4] which is useful for binary vectors a, etc. Any or all of these techniques may be used in our framework, but we focus on developing our proposals for sketches and demonstrating their potential.

- Finally, if s represents the array a, then `query(` i_1`,` i_2`,` s`)` returns a value in the range $(1 \pm \epsilon) \sum_{j=i_1}^{i_2} a_j$, except with a small failure probability δ, provided

  $$\frac{1}{i_2 - i_1 + 1} \left(\sum_{j=i_1}^{i_2} a_j \right)^2 \geq \tau \sum_j a_j^2;$$

  that is, provided the answer to the query is relatively large, depending on its length, and parameterized by $\tau < 1$. (It is always true that $\frac{1}{i_2 - i_1 + 1} \left(\sum_{j=i_1}^{i_2} a_j \right)^2 \leq \sum_j a_j^2$.)

If the result of a `cos()` or `query()` operation would be small and therfore unreliable, the sketch data structure reports that fact. It doesn't silently return unreliable information except as parametrized by δ and ϵ. All operations take time at most

$$(\log(N)/(\epsilon\tau))^{O(1)} \log(1/\delta).$$

The size of each sketch (which is invariant under `update` and `combine`) is also at most

$$(\log(N)/(\epsilon\tau))^{O(1)} \log(1/\delta),$$

using an implementation given in [13]. (Typically ϵ, δ, and τ are considered constant; in that case, the bound is polylogarithmic in N.) Thus, if N is very large, the size of the sketch is much smaller than N, the space needed for an exact array data structure. Note also that one can make the distortion, failure probability, and threshold as small as desired, though, as one would expect, the time and space required increases as these parameters decrease.

One can similarly sketch arrays of two or more dimensions. The `query` operation returns the sum of values in a specified rectangle.

An output answer may be accompanied by error bars (that depend on the prescribed parameters ϵ and τ and on the actual returned answer). We have observed, however, that the data structure often performs better in practice than the theoretical guaranteed worst case performance (which is not, after all, a prediction of performance). Since there are some theoretical guarantees, we suspect that more useful heuristic error bars would be well-behaved and can be determined through an experimental process.

3.3 Properties of Sketches

Several properties of sketches make them useful in distributed network data analysis. We now discuss this in more detail.

3.3.1 Basic Usage

An array_sketch is useful for consuming unaggregated feeds of data and answering queries about the aggregate. For example, consider a packet monitor that observes packets on a link, discarding all information except the source IP address and the number of bytes. We wish to make queries about the array a, such that a_i is the total number of bytes sent from IP address i. First, call `initialize()` to get a sketch s of the zero array. Each time we see a packet from i with b bytes, call $s \leftarrow$`update(` s, i, b`)`. At any stage, one can use the `query()` or `norm()` operations to find out about a. All operations are fast; only s, which is small, is stored. If a sketch needs to be shipped across the network, the small size of a sketch saves bandwidth.

3.3.2 Linearity of Sketches and the `combine` Operation

The `combine` operation facilitates distributed data collection. Suppose we are interested in the array of total traffic that exits our network via two gateway routers, G and G'. (For example, consider the array a, indexed by source IP address i, and such that each value a_i is the number of bytes transmitted by IP address i.) Router G initializes a sketch and sends a copy to router G'. (The routers must share a small "sketch handle" that contains the parameters ϵ, τ, and δ, and, more importantly, some common random bits used for subsequent sketch operations.) Each router then constructs a sketch s or s' of its own traffic a or a', then G sends s to G', and, finally, G' combines the sketches to get a sketch of $a + a'$. Without the `combine` operation (or a similar operation), to build a synopsis, data would have to be collected at a single place or raw data would have to be shipped to a single place.

3.3.3 Losslessness of `update` and `combine`

A sketch is built from an array via updates, which is useful if the array is not given all at once. For example, consider the array a indexed by IP address such that each value a_i contains the total number of bytes transmitted by i and passing a data collection point C on a particular day. Typically, as discussed in 3.3.1, the traffic sent by i is not sent all at once, so C never sees a_i. Instead, C typically needs to call `update` with the index i each of the many times it sees traffic from i. The update function is lossless in the sense that the same sketch is produced to describe the array a, no matter the order in which traffic is observed. Similarly, the `combine` operation is also lossless. That is, if we form sketch s for array a by a sequence of updates, then form sketch s' for array a' by another sequence of updates, then combine s and s', we'd get the same sketch for $a + a'$ as we'd get if apply both sequences of updates to an initialized sketch.

By contrast, some sampling techniques are lossy. Consider again the gateway routers G and G' that produce the arrays a and a' of total traffic in bytes, indexed by IP address. This time, we combine the arrays to get $a - a'$. Suppose we want to know by how much the traffic through G' to Sprint and UUNet exceeds the traffic through G to Sprint and UUNet. If this difference is a significant fraction of the total traffic, we will be able to estimate it. Note that this is the case even if we have the following situation:

	G	G'	$G'-G$
Sprint	100	4	
UUNet	2	103	
Sprint-UUNet Total	102	107	5
Rest of the net	200	210	10
All the net	302	317	15

The Sprint-UUNet traffic through G is 102 and the Sprint-UUNet traffic through G' is 107, so the difference is 5. The square 5^2 is a significant fraction of the overall number, $\sum_i (a_i - a'_i)^2 \le (317-302)^2 = 15^2$, so we will estimate 5 well, say, to within 10%. By contrast, some sampling-based techniques may ignore or miss G's UUNet traffic or G''s Sprint traffic, which would result in an error of 2 or 4 out of 5—in general, a worse approximation. Note that sketches are able to estimate 5 well even though sketches can *not* estimate well G's UUNet traffic of 2 or G''s Sprint traffic of 4, since G's UUNet traffic is insignificant compared with all of G's traffic and G''s Sprint traffic is insignificant compared with all of G''s traffic.

3.3.4 Adaptive Greedy Pursuit

The linearity of sketches enables a data analyst to perform *adaptive greedy pursuit*, using a tool borrowed from [12], where it was used in a different context. Suppose the array a is indexed by IP address and contains the number of transmitted bytes. Suppose further that $a_{i_{\text{CNN}}}$ traffic dominates the entire dataset, so that no other value is large enough to be estimated well from a sketch s for a. An analyst proceeds as follows. First, estimate $a_{i_{\text{CNN}}}$ from s, getting $\hat{a}_{i_{\text{CNN}}}$. Next, call $s' \leftarrow$`update(` s`,` i_{CNN}`,` $-\hat{a}_{i_{\text{CNN}}}$`)`. The result is a sketch of the array of traffic minus our (good) estimate of the CNN traffic. Any IP address i whose traffic is a significant part of the remaining non-CNN traffic (plus the small approximation error $a_{i_{\text{CNN}}} - \hat{a}_{i_{\text{CNN}}}$, that leaks into s'), can be estimated well from s'. In this case, many new IP addresses may be a significant fraction of the residue traffic, even if they were not a significant part of the total traffic. Note that we do not need to know in advance which is the dominant IP address (if we knew in advance, we could just record its traffic separately). Finally, note that greedy pursuit can be done using other sources of data. For example, suppose we construct sketches in near real time and get exact data for some IP address later. We can subtract off the available exact data from the sketch and use the resulting sketch to estimate the (now proportionally larger) other IP addresses.

3.4 Wavelets

An important use of array_sketch's is for wavelet analysis. A wavelet basis vector is a vector of length N of the form

$$\psi_{j,k} = (\overbrace{0,\ldots,0}^{k2^j}, \overbrace{-1,\ldots,-1}^{2^{j-1}}, \overbrace{1,\ldots,1}^{2^{j-1}}, 0,\ldots,0)/2^{j/2}.$$

Figure 4: The wavelet_sketch synopsis data structure.

```
w_sketch_t w_initialize( length_t N, distortion_t ε, threshold_t τ,
                         failure_prob_t δ, number_of_terms_t B);
w_sketch_t w_update( w_sketch_t s, index_t i, value_t v);
w_sketch_t w_combine( coefficient_t c1, w_sketch_t s1,
                      coefficient_t c2, w_sketch_t s2);
w_representation_t w_query( w_sketch_t s);
```

There are $N-1$ possible wavelet basis vectors; these, together with the vector $(1, 1, \ldots, 1)/\sqrt{N}$ form the orthonormal *wavelet basis* for the set of all vectors of length N. A wavelet query on a vector a by $\psi_{j,k}$ should return the dot product $\langle a, \psi_{j,k}\rangle$ (the *wavelet coefficient* $d_{j,k}$).

The vector a is recoverable from all its wavelet coefficients as $a = \sum_\ell \langle a, \psi_\ell\rangle \psi_\ell$. A lossy compression of the array a can be given by keeping only a subset Λ of the largest coefficients, $|\Lambda| = B \ll N$, and approximating a as $a \approx \sum_{\ell\in\Lambda} \langle a, \psi_\ell\rangle \psi_\ell$. The *energy* of this approximation is $\sum_{\ell\in\Lambda} \langle a, \psi_\ell\rangle^2$. Since $\sum_{\ell=1}^{N} \langle a, \psi_\ell\rangle^2 = \sum_{i=1}^{N} a_i^2$, the energy of a representation varies from zero (for the trivial zero approximation) to $\sum_{i=1}^{N} a_i^2$, for a perfect representation.

3.4.1 The Wavelet_Sketch Data Structure

A wavelet query is the difference of two range queries, so a wavelet query can be answered directly from the array_sketch data structure. A new operation, not quickly supported by the array_sketch data structure, is to find Λ, the best B-term wavelet representation. This can be done directly by estimating *all* N wavelet coefficients, but a quicker implementation requires additional data structures. The interface wavelet_sketch is summarized in Figure 4.

The operations `w_initialize`, `w_update`, and `w_combine` are similar to the array_sketch operations. The new operation, `w_query`, works as follows:

> If the array a is such that the best B-term wavelet approximation $R = \sum_{k=1}^{B} d_k \psi_k$ captures at least a τ fraction of the sum-square-norm, *i.e.*, $\sum_i R_i^2 \geq \tau \sum_i a_i^2$ or, equivalently, $\sum_i (a_i - R_i)^2 \leq (1-\tau) \sum_i a_i^2$, then, with probability $1-\delta$, the `w_query` operation returns a B-term wavelet representation $\hat{R}$ such that $\sum_i (a_i - \hat{R}_i)^2 \leq (1 - \tau + \epsilon\tau) \sum_i a_i^2$.

That is, the representation returned by `w_query` is, with high probability, at least $(1-\epsilon)$ times as good as the best B-term representation, provided the best B-term representation is pretty good, parameterized by τ. The time and space needed by the wavelet_sketch operations, using an implementation in [12], are similar to the array_sketch costs, but depend also on B:

$$(B \log(N)/(\epsilon\tau))^{O(1)} \log(1/\delta).$$

3.4.2 Using Wavelets

Information in Wavelets Themselves First, if the array a being sketched has a good B-term wavelet representation, the representation itself may provide information about a that the analyst did not think to ask –for example, there are several wavelet-based statistical tools for (multi)fractal traffic analysis [1, 14].

Finding Dominant Values If there is a single i_0 such that $a_{i_0}^2 \geq \eta \sum_i a_i^2$, then, from the best B-term wavelet representation, we can quickly find i_0. Similarly, we can quickly find a small number of such spikes. These results follow from known properties of wavelets [6].

Using Wavelets for Range Queries If one believes that a has a good B-term wavelet representation, it might even be advantageous to answer a range query (i_1, i_2) by calling `w_query` to get an approximation $\hat{R}$ to the best wavelet representation $R = \sum_{k=1}^{B} d_k \psi_k$ and then (quickly) answering the range query from $\hat{R}$ instead of from a. There are two advantages. First, $\sum_i R_i^2$ may be much larger than $\frac{1}{i_2 - i_1 + 1}\left(\sum_{i=i_1}^{i_2} a_i\right)^2$; in that case, our estimate of R will be much more reliable than our direct estimate of $\sum_{i=i_1}^{i_2} a_i$. If R and $\hat{R}$ are good representations of a, then the loss suffered by answering the query from R instead of a plus the loss in our approximation $\hat{R}$ to R may be less than the loss in our direct estimate of $\sum_{i=i_1}^{i_2} a_i$. Second, in many applications, the best wavelet representation (which has large coefficients) contains the "signal" and the remaining small coefficients contain "noise." In that case, R may be a better representation than a itself of the "true" phenomenon recorded noisily by a.

Data Visualization A wavelet representation for a is an example of a piecewise constant representation for a, which is relatively straightforward to render in a way sensible to humans. There is also a weak converse—a vector with a good piecewise constant representation R also has a good wavelet representation R' with not many more terms than there are pieces in R.

3.5 Using Sketches for Network Data

To use a sketch to summarize an array, a traffic engineer needs to decide how the array should be indexed and what values to store in the array. For example, the array might be indexed by time period, source or destination IP address or port, etc; the values stored may be the number of packets or bytes observed.

It is not significantly more expensive to index on both source and destination IP address, rather than on either alone. This is because, as will be explained, the cost depends on the logarithm of the number of indices, which increases only by a factor of two from all 2^{32} source addresses to all 2^{64} pairs of source and destination. It is important to emphasize, however, that one can only recover the significant information, a concept we formalize later.

The number of bytes from source i_s to destination i_d may be insignificant compared with all the traffic, while the number of bytes from i_s to any destination may be significant.

An update may be performed upon seeing each packet, or the raw data can be preprocessed before it is sketched, using any of the existing techniques. For example, one could filter packets and sketch only those coming from a particular subnet. One could also use the Cisco NetFlow facility to aggregate temporally and semantically connected packets, and sketch that data—*e.g.*, a_i is the number of flows from IP address i.

A network engineer must decide where to collect data—on a backbone router, gateway router, or access router. Because (a) sketches are small to store and to transmit, and (b) sketches produced at different routers can be combined readily and flexibly, it may be sensible to collect data at all routers. It is also possible to incrementally deploy collectors to one router at a time to produce useful data.

Our sketching technique can be seen as a special query that runs at the data collection server and is potentially useful for many different types of analysis. There are several advantages of using sketches to filter the data:

- Sketches can be made small enough to fit in various computing platforms, with provable quality guarantees. Storage and processing requirements are fixed given the size of the sketch and no additional scratch space is required.
- Sketches provide a compact description of the measured distribution. Thus, they are relatively cheap to communicate during distributed query execution.
- There are numerous transformations that can be reliably computed from the sketch with provable error guarantees. The wavelet transformation is such an example. Our experiments show that wavelet coefficient provide a very accurate description of the traffic on the domain of IP addresses. Other examples include similarity tests on various traffic vectors that can be used for clustering, setting up alarms etc.
- One more advantage of sketches over other compiled queries is that the single sketch query can be optimized well (hardware or software).

Sketches render the measured data in a format that allows us to mine for strong trends, as long as such trends exists. For example, looking at the wavelet transform of the traffic we can argue about self-similarity. Similarly, we can report queries that stand out of the signal, like heavy-hitters. On the contrary, we don't expect to accurately report statistics on blurred parts of the distribution. For instance the direct estimate of the traffic from an arbitrary IP address can be arbitrarily bad, when the answer is not significant with respect to the rest of the traffic. Such type of queries should be executed as a direct computation at the NIC or at the Data Warehouse.

4 Experiments

4.1 Set up

For the experiments presented in this section, we used NetFlow data obtain on three gateway routers of AT&T's backbone located in three major cities in the U.S. We refer to these routers as $router_1$, $router_2$ and $router_3$ respectively. There is a direct backbone link between $router_1 - router_3$ and $router_2 - router_3$ but not a direct link between $router_1$ and $router_2$.

From each router we obtained NetFlow records from a period of two hours, broken down into 12 intervals, each of which covers, roughly, 10 minutes. We denote each datafile as $F_{i,j}$, where index $i = 1, 2, 3$ refers to the router and $1 \leq j \leq 12$ denotes the time-period. The overall number of records (flows) in these files was roughly 30M (as presented in the cash register format; i.e., unaggregated and in chronological order). The NetFlow data is in a raw binary form, where records have information about the particular "engine" (e.g., which interface) that measured the flow and sequence numbers for the flow records observed by these engines. Some of the flow records are lost en route from the router to the collection server. We can detect such losses, via missing sequence numbers. On the dataset used, the loss factor was about 2% for all individual datafiles.

NetFlow records give a traffic engineer several options on how to index the flows and what values to store. For our experiments we indexed on either the source or the destination IP address. Unless otherwise noted, we aggregated on the number of bytes sent, thus a_i is the number of bytes sent (or received when destination address is used) from IP address i. The 30M flows yield 772318 distinct source and 1328997 distinct destination addresses.

In Subsection 4.2 we compute the wavelet decomposition for the available NetFlow data and show that network statistics compress well using wavelets. This is a strong result that justifies the use of sketches. Using sketches, we expect to capture reliably the large wavelet coefficients of the distribution and thus the overall trends of the traffic. Furthermore, small, fine-grained wavelet coefficients typically capture the inherent noise in the traffic. Thus, by eliminating those and using the large coefficients retained by the sketch we obtain better point-wise estimates of the traffic.

In subsection 4.3 we explore ways to analyze the datasets given that good summary information of the NetFlow traffic can be maintained using sketches. For the datasets that we tested we found that traffic patterns are quite dissimilar among different routers, but strongly related on the same router when examined at different time-periods. This result suggests that is possible to implement various clustering algorithms and use them for on-line classification of flows, setting up alarms, etc.

Finally, in subsection 4.4 we evaluate the estimates obtained from the sketches against real traffic measurements. Our results verify that strong patterns (like heavy hitters, large wavelet coefficients and cosines) can be detected and used for traffic analysis.

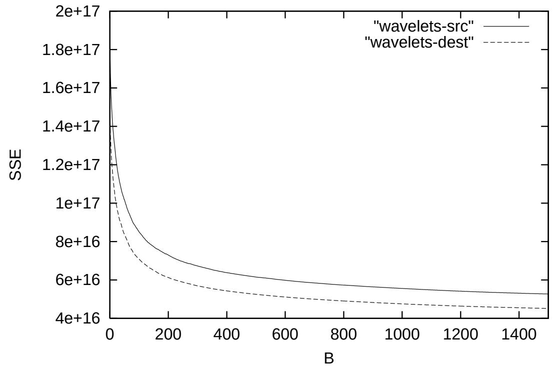

Figure 5: Decay of SSE with number of Coefficients Maintained

4.2 Decay of Wavelet coefficients on NetFlow Records

Wavelet transforms attempt to capture the trends in numerical functions. Often, very few of the wavelet coefficients are significant and thus, a small number of them can be used as a compact, accurate description of the function. This property makes wavelets appealing for compressing empirical datasets like images or call detail records [13].

In practice we evaluate how an empirical dataset "compresses" using wavelets by measuring the sum-squared-error (sse) obtained from an approximation R of array a using the largest B wavelet coefficients varying B. The rate of decay of sse with increasing B is a good indication of the applicability of wavelet methodology for a particular dataset.

In Figure 5 we plot the sse of a B-term wavelet representation of array a for the 1.4M NetFlow records of datafile $F_{1,1}$, when array a is indexed on source and destination IP address.[2] The number of distinct IP addresses in this dataset is 118,431. The graph shows a rapid decay of the error, as B increases. Notice that the x-axis interval of the Figure represents a very small fraction of the wavelet coefficients obtained. For the particular datafile, there are about 700,000 non-zero coefficients (out of 2^{32}=$4,294,967,296$ possible coefficients). In both cases, 500 coefficients (4Kbytes) retain about 70% of the energy of the array and thus result in a equal decrease in the sse. We observe that the network traffic, when indexed by destination IP address, has a more rapid decay and thus compresses better with the same number of coefficients. This is probably related to the different characteristics of the distribution of unique source and destination IP addresses.

4.3 Similarity Tests

The ability to use sketches to estimate the traffic array a opens new ways of analyzing the flows. Recall that sketches could be used to approximate dot products of vectors, as long as the result is large. In information retrieval, cosines between vectors that describe documents

[2] Other datafiles had similar graphs.

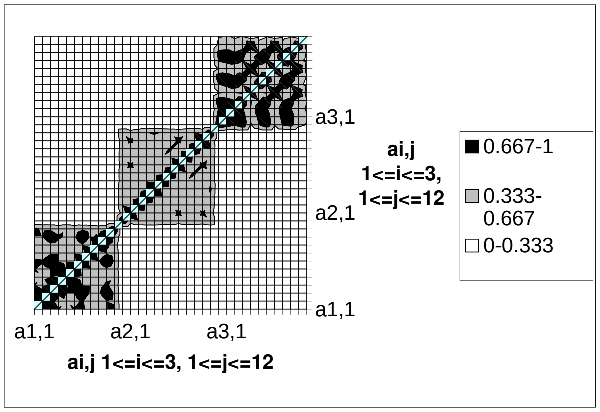

Figure 6: Similarity Matrix Based on cosines

in some feature space are frequently used to measure similarity among two documents. Such cosines are actually the dot-products of the corresponding normalized vectors. Using the same principal we can measure similarity of various traffic measurements along multiple routers and time periods. Given arrays a and b, indexed on the same domain (e.g. source IP) the cosine of the two vectors: $cos(a, b) = \langle a, b\rangle/\|a\|_2\|b\|_2$ can be approximated using their sketches as explained in Section 3. Thus, similarities can be detected reliably when they exist.

There are numerous ways to use sketches for such a computation. Some of those are listed bellow:

- **Compare traffic patterns along a timeline**. We can use the cosine-metric to test the hypothesis that traffic patterns on a router are similar along a timeline for a given measure (e.g. number of packets). For example, let $a^{t_1}, a^{t_2}, \ldots, a^{t_k}$ be the traffic vectors obtained on the router for time periods $t_1, \ldots t_n$. These can be consecutive hours/days etc. Using sketches of these vectors we can populate a $k \times k$ matrix with cosines of pairs a^{t_i}, a^{t_j} and identify periods with similar traffic.

- **Compare traffic over multiple routers**. We can use the cosine test to hypothesize on the amount of traffic that is sent between two routers. Similarly, we can test whether traffic is similar due to routing/topology invariants. For example, if a large ISP has peering links with two routers we can expect similarities on the traffic vectors obtained from these routers, as long as traffic from that ISP is a significant portion of the overall traffic in these routers.

- **Characterize traffic patterns using multiple measures**. Let a_{bytes} and $a_{packets}$ be the traffic vectors obtained from a router, counting bytes/packets sent from each source IP address. We expect the cosine of these two vectors to be large, when there is small deviation in the sizes of the packets sent from each source. Thus, we can use these vectors (or projections of then into appropriate subnets) to characterize sources (or destination) IPs. For example, traffic generated from ftp servers should be quite dissimilar to traffic due to DNS requests—the latter uses much smaller packet sizes.

We evaluated the cosine metric using arrays $a^{i,j}$ obtained from datafiles $F_{i,j}$ indexed by source-IP address (using the 24bit prefix). Thus, $a^{i,j}_{ip}$ measures the number of bytes received from 24bit subnet ip at router i during 10-minute interval j. We computed the cosines along all pairs of vectors and plotted the result in Figure 6. The x- and y-axis of the Figure represent vectors $a^{i,j}$. Dark areas reveal pairs with large cosine values. The following observations are made:

- traffic among different routers shows small similarities (based on source-IP and number of bytes sent). The same pattern was observed when looking at the destination address, or when using a binary traffic array: $a_i = true$ iff there has been a flow from (to) IP address i. We do not report these experiments here due to space limitations.
- traffic on the same router, along the measured two-hour interval is quite similar, especially for router $router_1$ and $router_3$. Router $router_2$ has more deviation on its traffic along the given timeline.

These results also suggest that we may use the cosine test to cluster flows and detect abnormal behavior.

4.4 Using Sketches for Finding Heavy Hitters

In this set of experiments we evaluate using sketches to locate source (destination) IP addresses that generate a significant proportion of the traffic. The basic query that we consider here is to locate the top-k (for some small value of k) source IP addresses on a given router based on the number of bytes sent. Let $L = \{ip_1, ip_2, \ldots, ip_k\}$ be a list of the top-k source 32bit addresses based on their traffic, *i.e.*, $a_{ip_1} \geq a_{ip_2} \geq \cdots \geq a_{ip_k}$. By sketching on source and bytes sent and using function `w_query` we obtain an approximation $\hat{R}$ of the traffic array. From that we reconstruct a list of heavy-hitters $\hat{L} = \{\hat{ip}_1, \hat{ip}_2, \ldots, \hat{ip}_k\}$ that is an approximation of L. We can then evaluate the precision of the sketch by comparing the two lists. A naive approach is to count the number of common elements on L and $\hat{L}$. This however would be misleading as an approximation that captures the first few-elements of L is more valuable than one that captures most of the trailing sources, especially in a long-tailed distribution. To capture that effect we evaluate list $\hat{L}$ by summing the energy of the elements in $\hat{L}$:$energy(\hat{L}) = \sum_{ip_j \in \hat{L}} a^2_{ip_j}$. By definition $energy(\hat{L}) \leq energy(L)$ and therefore the ratio $\frac{energy(\hat{L})}{energy(L)}$ is less than or equal to 1 (when $\hat{L} = L$).

We used dataset $F_{1,1}$ from the first router. The dataset contains 1.4M flows generated from 118,431 distinct IP-sources. We sketched these records using a sketch of size 7228

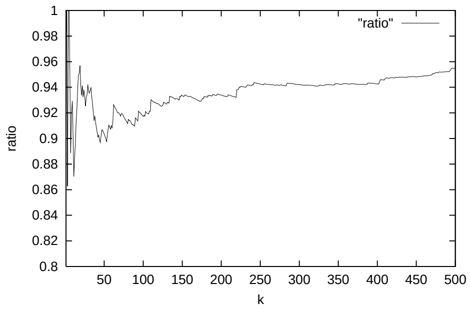

Figure 7: Ratio of energy in approximate top-k list over the energy of the exact answer

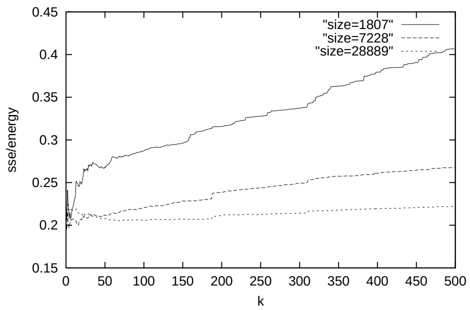

Figure 8: Ratio of sse in approximate top-k list over the energy of the exact answer for three sketch-sizes

(approximately 925Kbytes). For computing list $\hat{L}$, we reconstructed the traffic for the top-1000 sources (that we knew from the dataset), using the sketch, and assumed the estimate for the traffic of the remaining sources was zero. This was a shortcut made to decrease the computation time of our simulation and we don't expect it to affect the results we report here for $k < 1000$ because of the long-tailed distribution of the traffic.

Figure 7 plots the ratio of the energy of the approximate list $\hat{L}$ varying k between 1 and 500. An interpretation of this graph is that the sketch suggests a list of IPs that contain at least 86% of the traffic (in L_2-fashion) of the exact answer. The ratio is relatively steady with k exceeding 100 as the distribution is indeed long-tailed and there is no real distinction (traffic-wise) between IPs in the back of the two lists.

Figure 7 reflects the ability to order and report source addresses based on their approximate traffic calculated from the sketch, however it does not fully characterize the approximation obtained. For the latter we use the sum-squared-error (sse) of approximating the values in L: $sse(L) = \sum_{ip_j \in L}(a_{ip_j} - \hat{R}_{ip_j})^2$.

Figure 8 plots the sse in approximating the top-k heavy-hitters (varying k) over their energy (i.e. $sse(L)/energy(L)$) for three different sketch-sizes 1807, 7228 and 28899 (230Kbytes, 925Kbytes, and 3.7Mbytes, respectively). As expected, longer sketches provide better estimation, especially on the tails of the distribution.

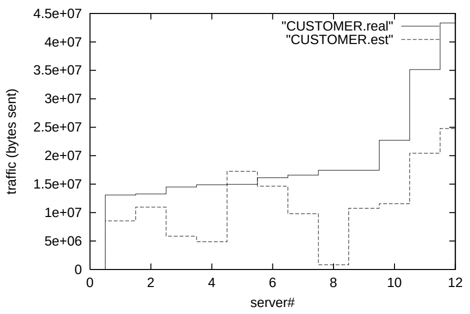

Figure 9: CUSTOMER traffic estimates

Among the top-100 heavy-hitters for this dataset, we found 12 servers for a single AT&T customer CUSTOMER. Figure 9 plots the real and estimated traffic sent from these servers to $router_1$. In general we do not expect to get a highly accurate description of array a on the 2^{32} domain of IP addresses, by using a sketch containing just 7228 values (925Kbytes). For reasonable sketch sizes, there will always be cases where traffic is egregiously miscalculated. We should evaluate the approximation in terms of the energy maintained by the sketch and not on individual point-wise estimates.

4.4.1 Analyzing Multiple Measures

In practice, we expect to maintain sketches on various metrics, which will allow more flexible analysis of the traffic. For example assume that we sketch on both the number of bytes and number of packets sent from each source using two sketches s_{bytes} and $s_{packets}$. Using these sketches we can answer queries that correlate both traffic measures.

For instance, we can compute sources with significant aggregate volume that sent few large packets (*e.g.*, ftp requests) or, similarly, sources that send many small packets (DNS servers, messages for routing protocols) of small aggregate size. One way to formulate the query is: "find sources that are in the top-k list with respect to the number of bytes sent but not in the top-k list based on the number of packets sent, and vise versa". Let L_{bytes}, $L_{packets}$ be the lists of sources based on bytes/packets sent respectively. We want to compute $L = L_{bytes} \bigcup L_{packets} - L_{bytes} \bigcap L_{packets}$. Using the sketches we obtain $\hat{L}_{bytes}$, $\hat{L}_{packets}$ and from these compute their symmetric difference $\hat{L}$. We now compare L and $\hat{L}$ using precision/recall:

$$recall = \frac{|L \bigcap \hat{L}|}{|L|} \quad precision = \frac{|L \bigcap \hat{L}|}{|\hat{L}|} \tag{1}$$

Figure 10 plots both metrics varying k on the previous dataset. For most cases, precision if around 30% and recall around 45%. This graph emphasizes the weaknesses of our techniques. In particular, we don't expect sketches to distinguish among sources with similar values in either of the lists. This is the reason for recall and precision being relatively low. On the other hand, using sketches we, most of the time, are able to obtain answers whose quality (energy-wise) is very close to the exact answer, see Figures 7, 8.

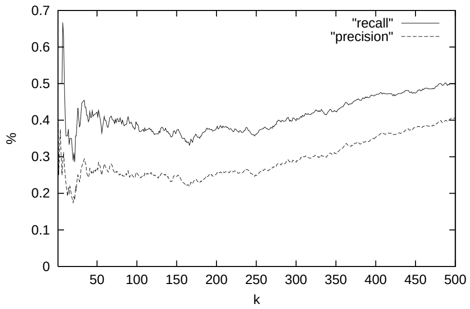

Figure 10: precision-recall measurements

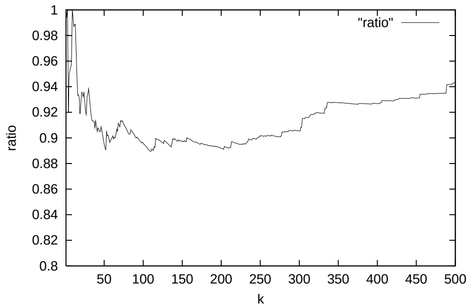

Figure 11: Ratio of energy in approximate top-k list over the energy of the exact answer for calculating change in traffic over two periods

4.4.2 Exploring linearity of sketches

Suppose we want to find sources whose traffic patterns have changed significantly between two time periods (*e.g.*, between two consecutive days). Given sketches $s^{i,j}$ and $s^{i,j'}$ of the traffic $a^{i,j}, a^{i,j'}$ during these periods we obtain $s = s^{i,j} - s^{i,j'}$. We can then estimate list L of the top-k hitters (in absolute terms) in the difference $a = a^{i,j} - a^{i,j'}$ as in the previous case. Providing the net-change in traffic from a particular IP address is significant with respect to the cumulative change between the two periods, it can be reliably computed from sketch s.

For the experiment of Figure 11 we computed list L of top-k sources with the larger net-change in traffic between vectors $a^{1,1}$ and $a^{1,2}$. The graph plots the energy of the approximate list $\hat{L}$ obtained by subtracting the corresponding sketches, over the energy of the exact answer L, varying k.

5 Conclusions

The challenges of data gathering and analysis in a large network seem to be surmountable only through the approximation. We have proposed using the sketch of the data to summarize it. Sketches use small space; they can be computed as data streams by, and can be combined across distributed sites. Sketches are able to respond well to queries that seek features (heavy hitters) that stand out of the data. They can also be used to generate compressed wavelet representation of the data and similarity measures. We supported our proposal by doing variety of experimental analysis based on sketches using AT&T WorldNet data. While our data collection and analysis infrastructure based on sketches is still preliminary, we believe that sketches are a promising research direction.

References

[1] P. Abry and D. Veitch. Wavelet analysis of long-range dependent traffic. *IEEE Transactions on Information Theory*, 44:2–15, 1998.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *ACM Symposium on Theory of Computing*, pages 20–29, 1996.

[3] R. Barquin and H. Edelstein, editors. *Building, Using and Managing the Data Warehouse*. The Data Warehousing Institute Series. Prentice Hall PTR, 1997.

[4] A. Z. Broder, M. Charikar, A .M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

[5] R. Caceres, N. Duffield, A. Feldmann, J. Friedmann, A. Greenberg, R. Greer, T. Johnson, C. Kalmanek, B. Krishnamurthy, D. Lavelle, P. Mishra, K. Ramakrishnan, J. Rexford, F. True, and J. van der Merwe. Measurement and Analysis of IP Network Usage and Behaviour. *IEEE Communications Magazine*, pages 144–151, May 2000.

[6] I. Daubechies. *Ten Lectures on Wavelets*, volume 61 of *Conference Board of the Mathematical Sciences*. SIAM, Philadelphia, 1992.

[7] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. pages 301–313. ACM SIGCOMM Conference, 1999.

[8] A. Feldmann, A. C. Gilbert, and W. Willinger. Data networks as cascades: Investigating the multifractal nature of Internet WAN traffic. In *Proc. of the ACM/SIGCOMM'98*, pages 25–38, Vancouver, B.C., 1998.

[9] A. Feldmann, A. C. Gilbert, W. Willinger, and T. G. Kurtz. The changing nature of network traffic: Scaling phenomena. *Computer Communication Review*, 28(2):5–29, 1998.

[10] A. Feldmann, A. G. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving Traffic Demands for Operational IP Networks: Methodology and Experience. In *ACM SIGCOMM'00*, pages 257–270, 2000.

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

[12] A. C. Gilbert, S. Guha, P. Indyk, I. Kotidis, S. Muthukrishnan, and M. J. Strauss. Untitled, 2001. Manuscript.

[13] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Surfing wavelets on streams: one-pass summaries for approximate aggregate queries. In *Proceedings of the 27'th International conference on Very Large Data Bases*, 2001. To appear.

[14] A. C. Gilbert, W. Willinger, and A. Feldmann. Scaling analysis of conservative cascades, with applications to network traffic. *IEEE Transactions on Information Theory*, 45:971–991, 1999.

[15] M. Grossglauser and J. Rexford. SIGCOMM 2001 Tutorial. 2001.

[16] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 41st Symposium on Foundations of Computer Science*, pages 189–197, Los Alamitos, CA, 2000. IEEE Computer Society.

[17] T. Johnson. personal communication.

[18] W. B. Johnson and J. Lindenstrauss. Extensions of lipshitz mapping into hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[19] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. In *Proc. of the ACM/SIGCOMM'00*, Stockholm, Sweden, 2000.

[20] P. Raghavan M. Henzinger and S. Rajagopalan. Computing on data streams. *SRC Technical Note*, 1998-011, 1998.