A Data Streaming Algorithm for Estimating Entropies of OD Flows

Haiquan (Chuck) Zhao
Georgia Inst. of Technology

Ashwin Lall
University of Rochester

Mitsunori Ogihara
University of Rochester

Oliver Spatscheck
AT&T Labs – Research

Jia Wang
AT&T Labs – Research

Jun (Jim) Xu[*]
Georgia Inst. of Technology

ABSTRACT

Entropy has recently gained considerable significance as an important metric for network measurement. Previous research has shown its utility in clustering traffic and detecting traffic anomalies. While measuring the entropy of the traffic observed at a single point has already been studied, an interesting open problem is to measure the entropy of the traffic between every origin-destination pair. In this paper, we propose the first solution to this challenging problem. Our sketch builds upon and extends the L_p *sketch* of Indyk with significant additional innovations. We present calculations showing that our data streaming algorithm is feasible for high link speeds using commodity CPU/memory at a reasonable cost. Our algorithm is shown to be very accurate in practice via simulations, using traffic traces collected at a tier-1 ISP backbone link.

Categories and Subject Descriptors

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

General Terms

Algorithms, Measurement, Theory

Keywords

Network Measurement, Entropy Estimation, Data Streaming, Traffic Matrix, Stable Distributions

1. INTRODUCTION

The (empirical) entropy of the network traffic has recently been proposed, in many different contexts, as an effective and reliable metric for anomaly detection and network diagnosis [3, 11, 15, 24, 25]. Measuring this quantity exactly in real time (say by maintaining per-flow states), however, is not possible on high-speed links due to its prohibitively high computational and memory requirements. For this reason, various data streaming algorithms [16] have been proposed to approximate this quantity. Data streaming [19] is concerned with processing a long stream of data items in one pass using a small working memory (called a sketch) in order to answer a class of queries regarding the stream. The challenge is to use this sketch to "remember" as much information *pertinent to our problem* as possible.

We observe that it is often important to know the entropies of origin-destination (OD) flows, where an OD flow is defined as all the traffic that enters an ingress point (called origin) and exits at an egress point (called destination). Knowing these quantities will give us significantly more insight into the dynamics of traffic inside an ISP network, which we will elaborate upon shortly. However, we found that none of the existing entropy estimation algorithms [1, 2, 5, 16] can be extended to solve this new problem. In particular, the traffic matrix estimation literature [18, 21, 22, 23, 27] has ruled out the following naive approach: separate the traffic at each ingress point into various OD flows in real time and feed them to existing entropy estimation algorithms. In this paper, we propose the first solution to this new problem.

1.1 Motivation

The ability to estimate entropy for all OD flows in a network can be highly beneficial. On today's Internet, network performance degradation and service disruption can be caused by a wide range of events, including anomalies (e.g., DDoS attacks, network failures, and flash crowds) and scheduled network maintenance tasks (e.g., router IOS updates and customer migration). Many of these events occur in a distributed manner (intentionally or unintentionally) in terms of their signatures and impacts. Detecting these events and evaluating their impact on network services thus often requires monitoring of the traffic from a number of locations across the network. More importantly, these changes in traffic distribution may be totally invisible in the traditional traffic volume matrix. However, by examining the entropy of every OD flow in the network, one can hope to capture these events in real time.

[*]Supported in part by NSF grants CNS 0716423, CNS 0626979, CNS 0519745, and NSF CAREER award CNS 0238315.

IMC'07, October 24-26, 2007, San Diego, California, USA.
Copyright 2007 ACM 978-1-59593-908-1/07/0010 ...$5.00.

To illustrate how beneficial the OD flow entropy estimation is, consider a situation in which a link fails (e.g., due to hardware problems) and is cost out for maintenance. The traffic flows that used to be carried on that link will most likely be redistributed to alternative paths. In the very rare case where no alternative path exists or the network experiences long convergence delay, the flows may abort completely. In either case, traffic distribution on a number of links across the network will change. It is possible that the traffic volume changes on those impacted links are small, and sometimes too small to be detected on each link using volume based detection algorithms. However, the overall change in the traffic distribution across the network can be quite large. If so, the examination of traffic entropy along with the traffic matrix for all OD flows is likely to enable us to capture such distributed and dynamically occurring events.

Another example is a DoS attack. When there is a distributed DoS attack launched from outside the network, its impact may not be immediately visible in the traffic matrix in terms of traffic volume change. However, it is very important for operators to be able to detect and mitigate the attack at the earliest time to minimize the possible negative impact on their services. That is, it may be too late for certain types of events to be detected when their impact becomes visible in the traffic volume matrix. In addition, certain types of DoS attacks are difficult to detect due to the low-rate nature of attack flows [14, 26]. In such cases, we hope to be able to achieve real time detection based on traffic entropy estimation.

1.2 Our solution and contributions

In this work, we propose the first solution to the challenging problem of estimating the entropies of all OD flows. Our key innovation is to invent a sketch that allows us to approximately estimate not only (a) the entropy of a data stream, but also (b) the entropy of the intersection of two data streams A and B from the sketches of A and B. We refer to property (b) as the "intersection measurable property" (IMP) in the sequel. Note that all existing entropy estimation algorithms (their sketches) have property (a), but none of them has IMP.

The intersection measurable property (IMP) of our sketch leads to the following very elegant solution of our problem. Each *participating* ingress and egress node in the network maintains a sketch of the traffic that flows in/out of it during a measurement interval. When we would like to measure the entropy of an OD flow stream between an ingress point i and an egress point j, we simply summon from node i the sketch of its ingress stream O_i, and from node j the sketch of its egress stream D_j, and then infer the entropy of $O_i \cap D_j$ using the sketches' IMP. This solution is extremely cost-effective in that we will be able to estimate $O(k^2)$ quantities (the entropies of a k by k OD flow stream matrix) using only $O(k)$ sketches. However sketches that have IMP are much harder to design than those without it, and hence the significant challenge we have in this work.

Our sketch builds upon and extends the L_p *sketch* of Indyk [12] with significant additional innovations. Indyk's sketch was designed for the estimation of the L_p norm of a stream (defined later). We observe that Indyk's L_p sketch has the desirable IMP, and therefore allows for the estimation of the L_p norm of an OD flow $O_i \cap D_j$ using the L_p sketches at O_i and D_j. However, we have to develop the IMP theory of L_p sketch ourselves since IMP was never even claimed in [13] (probably because there was no application in sight for IMP at that time).

Our most important contribution is to discover that the entropy of an OD flow can be approximated by a function of just two L_p norms (L_{p_1} and L_{p_2}, $p_1 \neq p_2$) of the OD flow. With this insight, our sketch at each ingress and egress point is simply one L_{p_1} sketch and one L_{p_2} sketch. In this way, we not only solve this difficult problem, but also find a nice application for the L_p sketch. From a theoretical point of view, our solution resolves one of the open questions of Cormode [7].

Our contributions can be summarized as follows. First, our discovery builds a mathematical connection between our problem and Indyk's L_p norm estimation problem, leading to a highly cost-effective solution. Second, we extend Indyk's (ϵ, δ) analysis of L_p sketches [13] using asymptotic normality of order statistics, leading to much tighter (yet slightly less rigorous) accuracy bounds. We also modify Indyk's algorithm to make it run fast enough for processing traffic at very high-speed links (e.g., 10 million packets per second). Finally, we thoroughly evaluate the accuracy of our solution by simulating it on real-world packet traces collected at a tier-1 ISP backbone. We show that our algorithm delivers very accurate estimations of the OD flow entropies.

The remainder of this paper is organized as follows. In Section 2 we give a high-level overview of how the different components of our entire scheme works. In Section 3 through 6 we describe the major components of our algorithm.

- In Section 3 we introduce stable distributions and describe Indyk's algorithm to estimate the L_p norm using them.
- In Section 4 we show how to estimate entropy from L_p norm.
- In Section 5 we show the intersection measurable property (IMP) of the L_p norm algorithm.
- In Section 6 we modify the L_p norm algorithm to improve its accuracy under tight resource constraints.

In Section 7 we discuss the hardware implementation of our algorithm. In Section 8 we demonstrate the accuracy of our algorithm via simulations run on real-world data. In Section 9 we briefly survey previous work that is related to ours. Finally, we conclude in Section 10. The Appendix contains the proof of some of the theorems.

2. PROBLEM STATEMENT AND OVERVIEW

In this section, we describe precisely the problem of estimating the entropies of OD flows and offer an overview of our solution approach. As defined before in [16], given a stream S of packets that contains n (transport-layer) flows with sizes (number of packets) a_1, a_2, ..., a_n respectively, its empirical entropy $H(S)$ is defined as $-\sum_{i=1}^{n} \frac{a_i}{s} \log_2(\frac{a_i}{s})$, where $s = \sum_{i=1}^{n} a_i$ is the total number of packets in S.[1]

[1] Throughout this paper logarithms will be base e except in the definition of empirical entropy here. Since the logarithm base e and the logarithm base 2 are different by a constant multiplicative factor, it does not affect our analysis of relative errors.

We have shown before [16] that to estimate the empirical entropy of a stream S, it suffices to to estimate a related quantity called *entropy norm* $||S||_H$, defined as $\sum_i a_i \ln(a_i)$, since $H(S)$ can be rewritten as

$$\begin{aligned} H(S) &= -\sum_i \frac{a_i}{s} \log_2\left(\frac{a_i}{s}\right) \\ &= \log_2(e)[\ln(s) - \frac{1}{s}\sum_i a_i \ln(a_i)], \end{aligned}$$

and s is usually a known quantity.

The entropy of an OD flow stream OD_{ij} between an ingress point i and an egress point j, is defined as the entropy of their intersection, i.e., $H(OD_{ij}) \equiv H(O_i \bigcap D_j)$. To derive this entropy value from the above formula, however, we need to measure both the entropy norm $||O_i \bigcap D_j||_H$ and s in order to compute the entropy of $H(O_i \bigcap D_j)$. Note that in our case, s is the volume of the OD flow and is an *unknown* quantity that needs to be estimated/inferred separately. As described in Section 1.2 , our solution is to invent a sketch for estimating the entropy norm of a stream that has the intersection measurable property (IMP).

Our algorithm and sketch build upon Indyk's classical results on estimating the L_p norm of a stream using the theory of stable distributions, for values of p in $(0, 2]$. In [13], Indyk presents an algorithm for computing the L_p norm of a stream S. For a stream S that contains n flows of sizes $a_1, \ldots, a_n$, its L_p norm $||S||_p$ is defined as $(\sum_i |a_i|^p)^{1/p}$, so $||S||_p^p = \sum_i |a_i|^p$. We discover that the data streaming solution of Indyk has the aforementioned IMP. Interestingly, this nice intersection property was never claimed in [13] or anywhere else. We will develop an entropy estimation technique that fully takes advantage of this property and offer its rigorous analysis.

While the work of Indyk is very influential, the practical importance of being able to estimate the L_p norm for values other than 1 and 2 was never clear. In our work, the L_p norms for p values slightly above or below 1 play a crucial role as follows. On realizing that Indyk's algorithm can be extended for estimating the L_p norms of an OD flow, we came up with the wild conjecture that it is possible to approximate the function $x\ln(x)$ using a linear combination of a small number of functions in the family $\{x^p | p \in (0, 2]\}$. In other words, we conjectured that we can find parameters $c_1, \ldots, c_k \in \mathbb{R}$ and $p_1, \ldots, p_k \in (0, 2]$ such that $x\ln(x) \approx \sum_{j=1}^k c_j * x^{p_j}$. If this conjecture is true, then we will be able to estimate the entropy norm of an OD flow stream S as $\sum_{j=1}^k c_j * ||S||_{p_j}^{p_j}$, where $||S||_{p_j}$, $j = 1, \ldots, k$ can be estimated using our extension of Indyk's algorithm for stream intersection. We emphasize that it took a leap of faith for us to think in this direction, as most of the approximation schemes we encountered in mathematical literature are by linear combination of terms like x^j (approximation by polynomial) and $\sin(x)$ (Fourier expansion).

Our wild conjecture has been proven correct! To our amazement we found that by using a linear combination of *only two* functions in the family, in the form of $\frac{1}{2\alpha}(x^{1+\alpha} - x^{1-\alpha})$, we can approximate $x\ln(x)$ very closely for all x values in a large interval, e.g., $[1, 1000]$ or $[1, 5000]$. Here α is a tunable parameter that takes small values. For example, in Figure 1, we show how closely we can approximate $x\ln(x)$ using $10(x^{1.05} - x^{0.95})$ within the interval $[1, 1000]$. In other words, if all transport-layer flows in an OD flow S have less than 1000 packets, we can estimate the entropy norm of the OD flow as $10(||S||_{1.05}^{1.05} - ||S||_{0.95}^{0.95})$. It turns out that this "symmetry" of the exponents ($1 + \alpha$ and $1 - \alpha$) around 1 serves another very important purpose that we will describe shortly.

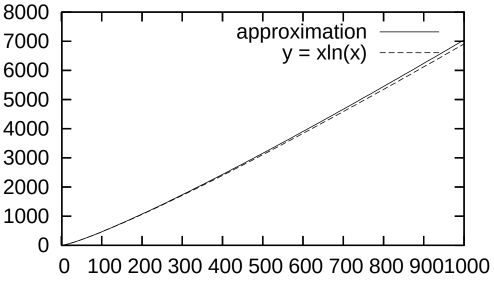

Figure 1: Comparison of entropy function and the approximation

We found that the approximation becomes gradually worse when x becomes larger than 1000 (packets), but the relative error level is generally acceptable up until 5000. Therefore, if there are some very large transport-layer flows (say with tens of or hundreds of thousands of packets) inside an OD flow S, the estimation formula such as $10(||S||_{1.05}^{1.05} - ||S||_{0.95}^{0.95})$ may deviate significantly from its entropy norm. Fortunately, identifying such very large flows is a well-studied problem in both computer networking (e.g., [10]) and theory (e.g., [17]). We will adopt the "sample and hold" algorithm proposed in [10] to identify all flows that are much larger than a certain threshold (say 1000 packets) and compute their contributions to the OD flow entropy separately.

Now the last piece of the puzzle is that when we compute the entropy H from the entropy norm, we need to know s, the total volume of the OD flow. Estimating this volume is a task known as traffic matrix estimation [18, 21, 22, 23, 27]. Various techniques for estimating the traffic matrix have been proposed that are based on statistical inference or direct measurement (including data streaming). In fact, this quantity is exactly the L_1 norm, which can be estimated using Indyk's L_1 norm estimation algorithm with IMP extensions. It turns out that we need none of these. Recall that our approximation formula is in the form of $\frac{1}{2\alpha}(x^{1+\alpha} - x^{1-\alpha})$, where α is a small number such as 0.05. We observe that, in this case, the function x can indeed be approximated by $(x^{1+\alpha} + x^{1-\alpha})/2$ and therefore the OD flow volume (i.e., the L_1 norm) can be approximated by the average of $||S||_{1+\alpha}^{1+\alpha}$ and $||S||_{1-\alpha}^{1-\alpha}$ calculated from the OD flow. Therefore, our sketch data structure allows us to kill two birds (the L_1 norm and the entropy norm) with one stone!

3. PRELIMINARIES

For the purposes of this paper we define a flow to be all the packets with the same five-tuple in their headers: source address, destination address, source port, destination port, and protocol.

Clearly, we will not be able to compute the entropy of each distribution exactly, so we use the following type of approximation scheme. An (ϵ, δ) approximation scheme is one that returns an approximation $\hat{\theta}$ for a value θ such that, with probability at least $1-\delta$, we have $(1-\epsilon)\theta \leq \hat{\theta} \leq (1+\epsilon)\theta$.

3.1 Stable distributions

The family of stable distributions, discovered first by Paul Lévy, is a natural generalization of the Gaussian distribution [29]. A stable distribution, in its full generality, takes a fairly complicated form with four parameters. In this work, however, we only need to work with the standard (normalized in a certain sense) cases that take only one free parameter p (the other three fixed), and the resulting restricted/standardized family is denoted $S(p), p \in (0, 2]$. Each $S(p)$ is uniquely characterized as follows. Let X be a random variable that takes distribution $S(p)$ with probability density function $f_{S(p)}(x)$. Then its characteristic function $E[e^{itX}]$ satisfies

$$E[e^{itX}] \equiv \int_{-\infty}^{\infty} f_{S(p)}(x)(\cos(xt) + i \cdot \sin(xt)) = e^{-|t|^p}.$$

The existence and uniqueness of a probability density function satisfying the above equation has been well established in mathematical literature.[2]

For any p, the probability density function $f_{S(p)}(x)$ of the distribution $S(p)$ is continuous and infinitely differentiable on $(-\infty, +\infty)$ (i.e., it is well-behaved). However, $f_{S(p)}(x)$ takes a closed form only for three p values ($p = 2, 1, 0.5$). $S(p)$ with $p = 2$ and 1 corresponds to two well-known examples of stable distributions: $S(2)$ is the Gaussian distribution with mean 0 and standard deviation 2, with probability density function $f(x) = \frac{1}{2\sqrt{2\pi}} e^{-x^2/8}$; $S(1)$ is the Cauchy distribution with the density function $f(x) = \frac{1}{\pi}\frac{1}{1+x^2}$. For p values other than 2, 1, and 0.5, we will not have a closed-form probability density function to work with. Instead, the distribution $S(p)$ is generated using a simulation program to be described later in Section 7. In the following, we state some important properties of the stable distribution that will be used in both Indyk's and our algorithms.

1. Stability property. The following property earns the name for the stable distributions. Let $X_1, X_2, \ldots, X_n$ denote mutually independent random variables that have distribution $S(p)$, then $a_1X_1 + a_2X_2 + \cdots + a_nX_n$ has the same distribution as $(|a_1|^p + |a_2|^p + \cdots + |a_n|^p)^{1/p}X$, where X is a random variable having distribution $S(p)$. It is not hard to verify that the aforementioned Gaussian and Cauchy random variables satisfy the stability property with parameter p equal to 2 and 1, respectively. It is also not hard to verify this property using the definition of $S(p)$ and the property of Fourier transforms.

2. Symmetry of the probability density function, $f_{S(p)}(x)$. For any $x \in (-\infty, \infty)$, we have $f_{S(p)}(x) = f_{S(p)}(-x)$. This can be verified from the fact that $f_{S(p)}(x)$ is the Fourier transform of $e^{-|t|^p}$.

3.2 Indyk's technique for estimating L_p norm

Estimating the L_p norm ($p \in (0, 2]$) of a stream using the property of stable distributions is a celebrated result in theoretical computer science [12, 13]. As defined before, given a stream S that contains n distinct objects with frequency a_1, a_2, ..., a_n respectively, the L_p norm of the stream is defined as $||S||_p \equiv (\sum_i |a_i|^p)^{1/p}$. Indyk's algorithm, translated into the language of TCP/IP, is shown in Algorithm 1.

Algorithm 1: Algorithm to compute L_p norm

```
Initialization
Let $Y[1 \ldots l]$ be an array of floating point numbers set
  to 0.0 at the beginning of measurement interval.
Fix $l$ p-stable hash functions $sh_1$ through $sh_l$.
Online stage
for each incoming packet $pkt$ do
    for $i := 1$ to $l$ do
        $Y[i]$ += $sh_i(pkt.id)$
Offline stage
Return $med(|Y[1]|, \ldots, |Y[l]|)/DMed_p$
```

The sketch is simply an array $\vec{Y}$ of real-valued counters set to 0.0 at the beginning of the measurement interval. The critical operator in this algorithm is a set of *p-stable hash functions* sh_i, $i = 1, ..., l$.[3] Each sh_i maps a flow label *pkt.id* into a random value drawn from the distribution $S(p)$, in such a way that (a) the same flow label will always be mapped to the same random value and (b) different flow labels are mapped to independent random values. In addition, these p-stable hash functions are independent of one another.

Online processing of packets with the sketch is also very simple. For each incoming packet, its flow label is hashed by sh_i and the result is added to Y_i. Now we analyze the value of a particular counter Y_1, and the analysis for other counters are exactly the same. Suppose that there are n flows with flow labels id_1, ..., id_n and of sizes a_1, ..., a_n. We denote $sh_1(id_1)$ as X_1, $sh_1(id_2)$ as X_2, ..., $sh_1(id_n)$ as X_n. By the property of sh_1, we know that X_1, ..., X_n are i.i.d. random variable with distribution $S(p)$. After processing the stream, we can see that the counter value Y_1 becomes $a_1 * X_1 + a_2 * X_2 + \cdots + a_n * X_n$. Since each X_i has distribution $S(p)$ and they are mutually independent, by the stability property, the counter value Y_1 is distributed as $(\sum_i |a_i|^p)^{1/p}X$, or in other words, $||S||_pX$, where X is distributed as $S(p)$. Therefore, counters Y_1, Y_2, ..., Y_l are i.i.d. draws from distribution $||S||_pX$. At this point, we can see that the data streaming algorithm is able to "modulate" the signal we would like to estimate, $(\sum_i |a_i|^p)^{1/p}$, into these counter values.

Now the question is how to extract this signal from these counter values? The approach proposed in [13] is to use the following median estimator for $||S||_p$:

$$\Lambda(\vec{Y}) \equiv \frac{\text{median}(|Y_1|, \ldots, |Y_l|)}{DMed_p}. \qquad (1)$$

We use $\Lambda(.)$ to denote the operator that extracts L_p norm estimates from sketches, and we will be using it in other contexts. Here $\text{median}(|Y_1|, \ldots, |Y_l|)$ denotes the sample median of the absolute values of the counter values; $DMed_p$ denotes the distribution median of $S^+(p)$. By $S^+(p)$ we mean the probability distribution of a random variable $|X|$, where X has distribution $S(p)$. We denote it as $S^+(p)$ because its p.d.f is extracly twice the positive half of the p.d.f for $S(p)$, due to the symmetry of $S(p)$. So $DMed_p$ is the unique (in this case) x_0 value such that $Pr[|X| > x_0] = 0.5$, where X

[2] In fact, in mathematical literature, the symbol α is often used in the place of p. We choose to use p to be consistent with [13], the most relevant related work.

[3] Note that our implementations of these hash functions are totally different than in [12, 13] to make them run much faster, which we will describe in Section 7.

has distribution $S(p)$. Note that, due to the symmetry of $S(p)$, $DMed_p$ is exactly the three-quarter quantile of $S(p)$. Although there is no closed form for $DMed_p$ for most of the p values, we can numerically calculate it by simulation, or we can use a program like [20].

Intuitively, the correctness of this estimator can be justified as follows. Since $Y_1/||S||_p$, ..., $Y_l/||S||_p$ are i.i.d. random variables with distribution $S(p)$, taking absolute value gives us i.i.d. draws from $S^+(p)$. For large enough l, their median should be close to the distribution median of $S^+(p)$. Therefore, we simply divide median$(|Y_1|, \ldots, |Y_l|)$ by the distribution median of $S^+(p)$ to get an estimator of $||S||_p$. We have to take absolute values because the distribution median of $S(p)$ is 0 due to its symmetry. In the next section we will analyze the relative error of this estimator.

Indyk's estimator for the L_p norm is based on the property of the median. We find, however, that it is possible to construct estimators based on other quantiles and they may even outperform the median estimator, in terms of estimation accuracy. However, since the improvement is marginal for our parameters settings, we stick to the median estimator.

3.3 Error analysis for L_p norm estimator

In this section we analyze the performance of the estimator for $||S||_p$ in (1).

3.3.1 (ϵ, δ) bound for $p = 1$

Here we basically restate Lemma 2 and Theorem 3 from Indyk [13] with the constants spelled out. We arrived at this by using Chernoff bounds to derive the constant in his Claim 2.

Theorem 1. *(Indyk, [13]) Let $\vec{X} = (X_1, \ldots, X_l)$ be i.i.d. samples from $S(1)$, $l = 8(\ln 2 + \ln \frac{1}{\delta})/\epsilon^2$, $\epsilon < 0.2$, then $DMed_1 = 1$, and $Pr[median(|X_1|, \ldots, |X_l|) \in [1-\epsilon, 1+\epsilon]] > 1 - \delta$. Thus (1) gives an (ϵ, δ) estimator for $p = 1$.*

Example: for $p = 1, \delta = 0.05$, $\epsilon = 0.1$, we get $l = 2951$. This is a very loose bound in the sense that we need a very large l. This motivates us to resort to the asymptotic normality of the median for some approximate analysis.

3.3.2 Asymptotic (ϵ, δ) bound for $p \in (0, 2]$

Theorem 2. *Let $f = f_{S^+(p)}$, $m = DMed_p$, $l = (\frac{z_{\delta/2}}{2mf(m)\epsilon})^2$, then (1) gives an estimator with asymptotic (ϵ, δ) bound. z_a is the number such that for standard normal distribution Z we have $Pr[Z > z_a] = a$.*

Proof is in the Appendix. This result is in the same order of $O(1/\epsilon^2)$ as the Chernoff result, but the coefficient is much smaller as shown below.

Example: For $p = 1, \delta = 0.05, \epsilon = 0.1$, we get $m = 1, f(m) = 1/\pi, z_{\delta/2} = 2, l = 986$. Compare with $l = 2951$ from the previous section.

For $p = 1.05$, we get $m = 0.9938, f(m) = 0.3324, l = 916$. For $p = 0.95$, we get $m = 1.0078, f(m) = 0.3030, l = 1072$. Our simulations show that these are quite accurate bounds. We can see that $mf(m)$ does not change much in a small neighborhood of $p = 1$. Since we are only interested in p in a small neighborhood of 1, for rough arguments we may use $mf(m)$ at $p = 1$, which is $1/\pi$.

4. SINGLE NODE ALGORITHM

In this section, we show how our sketch works for estimating the entropy of the traffic stream on a single link; Estimation of OD flow entropy based on its intersection measurable property (IMP) is the topic of the next section. We first show how to approximate the function $x \ln(x)$ by a linear combination of at most two functions of the form x^p, $p \in (0, 2]$. After that we analyze the combined error of this approximation and Indyk's algorithm.

4.1 Approximating $x \ln x$

Our algorithm computes the entropy of a stream of flows by approximating the entropy function $x \ln x$ by a linear combination of expressions x^p, $p \in (0, 2]$. In this section we demonstrate how to do this approximation up to arbitrary relative error ϵ. To make the formula simpler we use the natural logarithm $\ln x$ instead of $\log_2 x$, noting that changing the base is simply a matter of multiplying by the appropriate constant, thus having no effect on relative error.

Theorem 3. *For any $N > 1$, $\epsilon > 0$, there exists $\alpha \in (0, 1)$, $c = \frac{1}{2\alpha} \in O(\frac{\ln N}{\sqrt{\epsilon}})$, such that $f(x) = c(x^{1+\alpha} - x^{1-\alpha})$ approximates the entropy function $x \ln x$ for $x \in (1, N]$ within relative error bound ϵ, i.e., $|\frac{f(x) - x \ln x}{x \ln x}| \leq \epsilon$.*

Proof. Using the Taylor expansion,

$$x^\alpha = e^{\alpha \ln x} = 1 + \alpha \ln x + \frac{(\alpha \ln x)^2}{2!} + \frac{(\alpha \ln x)^3}{3!} + \cdots,$$

we get that

$$f(x) = x \ln x + \frac{\alpha^2 x \ln^3 x}{3!} + \frac{\alpha^4 x \ln^5 x}{5!} + \cdots$$

Rewriting in terms of the relative error, we get that

$$\begin{aligned} r(x, \alpha) \equiv \frac{f(x)}{x \ln x} - 1 &= \frac{(\alpha \ln x)^2}{3!} + \frac{(\alpha \ln x)^4}{5!} + \cdots \\ &= \sum_{k=1}^{\infty} \frac{(\alpha \ln x)^{2k}}{(2k+1)!}. \end{aligned}$$

Since every term is positive, we have $r(x, \alpha) \geq 0$. We assume that $\alpha < \frac{1}{\ln N}$. This gives us

$$r(x, \alpha) \leq \frac{1}{6} \sum_{k=1}^{\infty} (\alpha \ln x)^{2k} = \frac{1}{6} \left(\frac{(\alpha \ln x)^2}{1 - (\alpha \ln x)^2} \right). \quad (2)$$

The bound takes maximum value at $x = N$. Solving $\frac{1}{6}\left(\frac{(\alpha \ln N)^2}{1 - (\alpha \ln N)^2}\right) = \epsilon$ gives us $\alpha = \frac{\sqrt{\frac{6\epsilon}{1+6\epsilon}}}{\ln N}$, and $c = \frac{1}{2\alpha} = \frac{\ln N}{2\sqrt{\frac{6\epsilon}{1+6\epsilon}}} \in O(\frac{\ln N}{\sqrt{\epsilon}})$. Therefore $f(x) = \frac{1}{2\alpha}(x^{1+\alpha} - x^{1-\alpha})$ approximates $x \ln x$ within the relative error bound ϵ. □

A plot of this approximation for the range $[1, 1000]$ and $f(x) = 10(x^{1.05} - x^{0.95})$ is given in Figure 1. The relative error guarantee of the approximation only holds for values less than some constant N. As we have mentioned, we will use some elephant detection mechanism to circumvent this shortcoming.

4.2 Estimating entropy norm $||S||_H$

Now we will combine our approximation formula and Indyk's algorithm to get an estimator for the entropy norm $||S||_H$. Suppose we have chosen α and c in Theorem 3 to get relative error bound ϵ_0 on $[1, N]$, and we have chosen l

from Theorem 1 for $p = 1 \pm \alpha$ to achieve asymptotic (ϵ, δ) error bound. For $p = 1 + \alpha$ we have sketches $\vec{Y}$, and for $p = 1 - \alpha$ we have sketches $\vec{Z}$. Our estimator for $||S||_H$ is

$$\widehat{||S||_H} \equiv \frac{1}{2\alpha}\left(\Lambda(\vec{Y})^{1+\alpha} - \Lambda(\vec{Z})^{1-\alpha}\right) \qquad (3)$$

We now study the error of this estimator. We will use some approximation to get some rough but simple error estimates. The proofs are in the Appendix.

PROPOSITION 4. *Assume $a_i \leq N$. Then (3) estimates $||S||_H$ within relative error roughly $2\lambda c\epsilon + \lambda_0\epsilon_0$ with probability roughly $1 - 2\delta$, where $c = \frac{1}{2\alpha}$, $\lambda_0 = |c(||S||_{1+\alpha}^{1+\alpha} - ||S||_{1-\alpha}^{1-\alpha})/||S||_H - 1|/\epsilon_0 \leq 1$, $\lambda = ||S||_{1+\alpha}^{1+\alpha}/||S||_H$, and typically $\lambda < 1$.*

Example: For $N = 1024, \alpha = 0.05, \epsilon = 0.001, \delta = 0.05$, then $\epsilon_0 = 0.023, l \approx 10^5$. If we only assume $\lambda = \lambda_0 = 1$, then $(2c\epsilon + \epsilon_0) \approx 0.04$, i.e. we can approximate $||S||_H$ within 4% error with 90% probability using 10^5 samples. If we assume $\lambda = 0.5$, then we can afford to increase ϵ to 0.002, and thus decrease l to $\approx 2.5 \times 10^4$ to achieve the 4% error.

PROPOSITION 5. *(More Aggressive): Under same assumptions as above, (3) estimates $||S||_H$ within relative error roughly $\sqrt{2}c\lambda\epsilon + \lambda_0\epsilon_0$ with probability roughly $1 - \delta$.*

Example: Same assumption as the example above, then we only need $l \approx 1.25 \times 10^4$ to achieve the 4% error with probability 95%.

4.3 Estimating L_1 norm s

Recall that to compute the actual entropy $H = \log_2 s - \frac{1}{s}\sum_i a_i \log_2 a_i$, we need to know the complete volume of the traffic s, or $||S||_1$. For a single stream this is trivial to do with a single counter. But our ultimate goal is to calculate entropy of every OD pair, thus we need to compute the entire traffic matrix for the network. There has been considerable previous work in solving this problem, but any additional method will have the corresponding overhead associated with it. In this section we show how to use the same sketch data structure that we have been maintaining so far to approximate this value for a single stream, which will be naturally extended to distributed case later.

Similar to the proof of Theorem 3, we can easily show the following theorem:

THEOREM 6. *Let α, N, and ϵ be as in Theorem 3. Then, the approximation $(x^{1+\alpha} + x^{1-\alpha})/2$ approximates the function $f(x) = x$ with relative error at most 3ϵ.*

This approximation holds good for all counts in the range $[1, N]$ and we use the elephant sketch to capture all flows with size strictly greater than N.

So our estimator for $||S||_1$, is

$$\widehat{||S||_1} \equiv \frac{1}{2}\left(\Lambda(\vec{Y})^{1+\alpha} + \Lambda(\vec{Z})^{1-\alpha}\right). \qquad (4)$$

Using Theorem 6 and proofs similar to those of Propositions 4 and 5, we have the following:

PROPOSITION 7. *(4) estimates $||S||_1$ roughly within relative error bound $\epsilon + 3\lambda_0'\epsilon_0$ with probability $1 - 2\delta$, where $\lambda_0' = |(||S||_{1+\alpha}^{1+\alpha} + ||S||_{1-\alpha}^{1-\alpha})/||S||_1 - 1|/3\epsilon_0 \leq 1$. Or, more aggressively, the error bound is roughly $\frac{1}{\sqrt{2}}\epsilon + 3\lambda_0'\epsilon_0$ with probability $1 - \delta$.*

4.4 Separating elephants

Recall that we need to keep track of the elephant flows (say those that have more 1000 packets) and estimate their contributions to the total entropy separately. In our scheme, we adopt the "sample and hold" algorithm proposed in [10] due to its low computational complexity and ease of analysis. The "sample and hold" algorithm will produce a list of flows that include all elephant flows with very high probability. Then for each elephant flow in the list, we subtract the increments caused by them to the sketches, and compute their contributions to the entropy separately. The algorithm also has the nice property that the larger the size of a flow, the smaller (actually exponentially smaller) the probability of its missing from the list. This property works very well with the fact that the accuracy of our approximation of $x\ln(x)$ degrades only gradually after the target threshold (say 1000 packets).

5. DISTRIBUTED ALGORITHM

In this section we show the IMP property of the L_p sketch, i.e., we can use the sketches at ingress nodes and egress nodes to estimate the L_p norm of the OD flows.

For a given OD-pair, we will require only the sketches at that ingress and egress node. Hence, we fix one such pair and do all the analysis for it. We call the ingress stream as O and egress stream as D. We are interested in the flows in the set $O \cap D$. Note that if we can estimate the pair of L_p norms for $O \cap D$, $p = 1 \pm \alpha$, then we can use the formula for approximating the entropy function as before.

The sketch data structures will be the same at every ingress and egress node, that is, they will use the same number of counters l, and they will use the same set of *p-stable hash functions* as defined in Section 3.2. After we introduce bucketing in Section 6, they will also use the same number of buckets k and the same uniform hash function.

5.1 Computing OD flow L_p norm $||O \cap D||_p$

With a slight abuse of notation, we will use $\vec{O}$ to denote the sketch for the ingress node, and $\vec{D}$ to denote the sketch for the egress node. $\vec{O} + \vec{D}$ and $\vec{O} - \vec{D}$ are the componentwise addition and difference of $\vec{O}$ and $\vec{D}$ respectively. (This is possible because all nodes are using the same values of l.)

Our estimator for $||O \cap D||_p$, $||\widehat{O \cap D}||_p$, can be either

$$\Lambda(\vec{O}, \vec{D}) \equiv \left(\frac{\Lambda(\vec{O})^p + \Lambda(\vec{D})^p - \Lambda(\vec{O} - \vec{D})^p}{2}\right)^{1/p} \qquad (5)$$

or

$$\Lambda'(\vec{O}, \vec{D}) \equiv \left(\frac{\Lambda(\vec{O} + \vec{D})^p - \Lambda(\vec{O} - \vec{D})^p}{2^p}\right)^{1/p}. \qquad (6)$$

5.2 Correctness

In this section we show that the two formulae described are good estimators for $||O \cap D||_p$. Hence, by using two copies of the above algorithm, one each for $p_1 = 1 - \alpha$ and $p_2 = 1 + \alpha$, and our $x \ln x$ approximation formula, we can obtain an approximation of the entropy between every pair of ingress and egress nodes.

We partition the flows that enter through the ingress node or exit through the egress nodes as follows:

$A = O - D =$ flows that enter at ingress but do not exit through the egress
$B = O \cap D =$ flows that enter at ingress and exit through the egress
$C = D - O =$ flows that do not enter at ingress but exit through the egress

We know that $\Lambda(\vec{O})$ is an estimator for $||O||_p$, so $\Lambda(\vec{O})^p$ is an estimator for $||O||_p^p = ||A \cup B||_p^p = ||A||_p^p + ||B||_p^p$. Similarly $\Lambda(\vec{D})^p$ is an estimator for $||B||_p^p + ||C||_p^p$.

The sketch $\vec{O} + \vec{D}$ holds the contributions of all the flows in A, B and C, but with every packet from B contributing twice. We use $B^{(2)}$ to denote all the flows in B with packet counts doubled. Then $\Lambda(\vec{O} + \vec{D})^p$ is an estimator for $||A \cup B^{(2)} \cup C||_p^p = ||A||_p^p + 2^p ||B||_p^p + ||C||_p^p$. It is important to point out that the reasoning here (and in the next paragraph) depends on the fact that the ingress and egress nodes are using the same sketch settings as noted at the beginning of this section.

The sketch $\vec{O} - \vec{D}$ exactly cancels out the contributions from all the flows in B, and leaves us with the contributions of flows from A and the negative of the contributions of flows from C. We use $C^{(-1)}$ to denote all the flows in C with packet counts multiplied by -1. Then $\Lambda(\vec{O} - \vec{D})^p$ is an estimator of $||A \cup C^{(-1)}||_p^p = ||A||_p^p + ||C^{(-1)}||_p^p = ||A||_p^p + ||C^{|}|_p^p$.

To sum up, we get the following:

$$\begin{array}{rll} \Lambda(\vec{O})^p & \text{estimates} & ||A||_p^p + ||B||_p^p \\ \Lambda(\vec{D})^p & \text{estimates} & ||B||_p^p + ||C||_p^p \\ \Lambda(\vec{O} + \vec{D})^p & \text{estimates} & ||A||_p^p + 2^p ||B||_p^p + ||C||_p^p \\ \Lambda(\vec{O} - \vec{D})^p & \text{estimates} & ||A||_p^p + ||C||_p^p. \end{array}$$

It is easy to see from the above formulae that both Formula (5) and Formula (6) are reasonable estimators for $||B||_p$, i.e. $||O \cap D||_p$.

Proposition 8. *Suppose we have chosen proper l such that (1) is roughly an (ϵ, δ) estimator. Suppose $||O \cap D||_p^p = r_1 ||O||_p^p$, and $||O \cap D||_p^p = r_2 ||D||_p^p$. Then (5) raised to the power p estimates $||O \cap D||_p^p$ roughly within relative error bound $(\frac{1}{r_1} + \frac{1}{r_2} - 1)\epsilon$ with probability at least $1 - 3\delta$. Similarly, (6) raised to the power p estimates $||O \cap D||_p^p$ roughly within relative error bound $2^{1-p}(\frac{1}{r_1} + \frac{1}{r_2} + 2^{p-1} - 2)\epsilon \approx (\frac{1}{r_1} + \frac{1}{r_2} - 1)\epsilon$ with probability at least $1 - 2\delta$.*

We omit the proof here since it is similar to the previous proofs. This gives us a very loose rough upper bound on the relative error. The ratios r_1 and r_2 are related to, but not identical to, the ratio of OD flow traffic against the total traffic at the ingress and egress points. We want to point out that we cannot pursue a more aggressive claim similar to Proposition 5, because we cannot claim independence of $\Lambda(\vec{O})$ and $\Lambda(\vec{O} + \vec{D})$, etc.

Now, to calculate OD flow entropy, we just need to replace $\Lambda(\vec{Y})$ in (1) and (4) with $\Lambda(\vec{O}, \vec{D})$ where $\vec{O}$ and $\vec{D}$ are $L_{1+\alpha}$ sketches, and similarly for $\Lambda(\vec{Z})$.

6. USING BUCKETS

Our earlier examples showed that l, the number of counters in the L_p sketch, need to be in the order of many thousands to achieve a high estimation accuracy. Recall that each incoming packet will trigger increments to all l counters for estimating one L_p norm, and our algorithm requires that two different L_p norms be computed. Such a large l is unacceptable to networking applications, however, since for high-speed links, where each packet has tens of nanoseconds to process, it is impossible to increment that many counters per packet, even if they are all in fast SRAM.

We resolve this problem by adopting the standard methodology of bucketing [9], shown in Algorithm 2. With bucketing, the sketch data structure becomes a two-dimensional array M[1..k][1..l]. For each incoming packet, we hash its flow label using a uniform hash, function uh, and the result $uh(pkt.id)$ is the index of the bucket at which the packet should be processed. Then we increment the counters $M[uh(pkt.id)][1 \ldots l]$ like in Algorithm 1. Finally, we add up the L_p estimations from all these buckets to obtain our final estimate.

In the following, we will show that, roughly speaking, bucketing (i.e., with k buckets instead of 1) will reduce the standard deviation of our estimation of L_p norms by a factor slightly less than $\sqrt{k}$, provided that the number of flows is much larger than the number of buckets k. Lemma 9 shows that the standard deviation for using l counters is in the order of $O(1/\sqrt{l})$. Therefore, a decrease in l can be compensated by an increase in k by a slightly larger factor. In our proposed implementation (described in Section 7), the number of buckets k is typically on the order of 10,000. Such a large bucket size allows l to shrink to a small number such as 20 to achieve the same (or even better) estimation accuracy. We will show shortly that, even on very high-speed links (say 10M packets per second), a few tens of memory accesses per packet can be accommodated.

We use B_i to denote all the flows hashed to the ith bucket, and $||B_i||_p$ to denote the L_p norm of the flows in the ith bucket, similar to how we defined $||S||_p$ before. Obviously $||S||_p^p = \sum_i ||B_i||_p^p$. Let M_i be the i-th row vector of the sketch. We know that $\Lambda(M_i)$ as defined in (1) is an estimator of $||B_i||_p$, so naturally the estimator for $||S||p$ is:

$$\Lambda(M) \equiv \left[\sum_{i=1}^{k} (\Lambda(M_i))^p \right]^{1/p}. \tag{7}$$

In the ideal case of even distribution of flows into buckets, and all $||B_i||_p^p$ are the same, then $||S||_p^p = k||B_i||_p^p$. Let's consider $p = 1$ first. Lemma 9 tells us that the estimator $\Lambda(M_i)$ is roughly Gaussian with mean $||B_i||_1$ and standard deviation $(1/2mf(m)\sqrt{l})\|B_i\|_1$. By the Central Limit Theorem, $\Lambda(M) = \sum_{i=1}^{k} \Lambda(M_i)$ is asymptotically Gaussian, and its standard deviation is roughly $\sqrt{k}(1/2mf(m)\sqrt{l})\|B_1\|_1 = (1/2mf(m)\sqrt{lk})||S||_1$. If we didn't use any buckets and simply used estimator (1), then the standard deviation would be $(1/2mf(m)\sqrt{l})||S||_1$. So lk is performing the same role as l in the previous analysis, or in other words, k buckets reduces standard error roughly by a factor of $\sqrt{k}$.

When $p = 1 \pm \alpha$ in a small neighborhood of 1, we can reach the same conclusion by using the same handwaving argument in proof of Proposition 5 that a Gaussian raised to power p is still roughly Gaussian.

In reality, we will not have even distribution of flows into various buckets. However, when the number of flows is far larger than the number of buckets, which will be the case with our parameter settings and intended workload, we can

Algorithm 2: Algorithm to compute L_p norm with bucketing

```
Pre-processing stage
Initialize a sketch $M[1 \ldots k][1 \ldots l]$ to all zeroes
Fix $l$ p-stable hash functions $sh_1$ through $sh_l$.
Let hash function $uh$ map flow labels to $\{1, \ldots, k\}$
Calculate expected sample median $EMed_{p,l}$ by simulation
Online stage
for each incoming packet $pkt$ do
    for $j := 1$ to $l$ do
        $M[uh(pkt.id)][j]$ += $sh_j(pkt.id))$
Offline stage
Return $\left[\sum_{i=1}^{k} \left(\frac{\text{median}(|M[i][1]|,\ldots,|M[i][l]|)}{EMed_{p,l}}\right)^{p}\right]^{1/p}$
```

prove that the factor of error reduction is only slightly less than $\sqrt{k}$. We omit the proof here due to lack of space[4]. When k is increased to be on the same order of the number of flows, however, the factor of error reduction will no longer scale as $\sqrt{k}$ since (a) there will be many empty buckets that will not contribute to the reduction of estimation error, and (b) distribution of flows into buckets will be more and more uneven. Therefore, when l is fixed, the estimation error cannot be brought down arbitrarily close to 0 by increasing k arbitrarily.

Another issue is the bias of median estimator (1), that is, the expected value of the sample median of l samples is not equal to the distribution median $DMed_p$. When we are not using buckets, the asymptotic normality implies that the bias is much smaller than the standard error, so we could ignore the issue. Now that we are using k buckets to reduce the standard error by a factor of $\sqrt{k}$, the bias becomes significant. Let $EMed_{p,l}$ denote the expected value of the median of l samples from distribution $S^+(p)$. So we redefine our estimator for $||B_i||_p$:

$$\tilde{\Lambda}(M_i) \equiv \frac{\text{median}(|M[i][1]|, \ldots, |M[i][l]|)}{EMed_{p,l}}. \tag{8}$$

This replaces $\Lambda(M_i)$ in (7) and gives our estimator using buckets. Note that (8) is an unbiased estimator, but (7) still may be biased.

Let us assume that M and N are L_p sketches with buckets at ingress node O and egress node D respectively, and the two sketches use the same settings. We can replace $\vec{O}$ and $\vec{D}$ in (5) and (6) with M and N, and it is easy to repeat the arguments there to show that these are reasonable estimators for the OD-flow L_p norm.

$EMed_{p,l}$ can be numerically calculated using the p.d.f. formula for order statistics when $f_{S(p)}$ has closed form. Or it can be derived via simulation. We can also talk about $VMed_{p,l}$, variance of the sample median of l samples.

Example: For $p = 1$, $l = 20$, we get $EMed_{1,20} = 1.069$, $VMed_{1,20} = 0.149$, so standard deviation is 0.386. The distribution median is $DMed_1 = 1$, so we can see the bias 0.069 is much smaller than the standard deviation 0.386. Also the asymptotic standard deviation given by Lemma 9 is 0.351, which is close to the actual value of 0.385.

7. ALGORITHM IMPLEMENTATION

Our data streaming algorithm is designed to work with high link speeds of up to 10 million packets per second using commodity CPU/memory at a reasonable cost. In this section, we explain how various components of the algorithm shall be implemented to achieve this design objective.

Recall that our algorithm needs to keep two sub-sketches for estimating the $L_{1+\alpha}$ and the $L_{1-\alpha}$ norms respectively, each of which consists of $k * l$ real-valued counters. We set the number of counters per bucket l to 20 in our proposed implementation. Since the sketches are implemented using inexpensive high-throughput DRAM (explained next), we allow the number of buckets k to be very large (say up to millions).

As shown in Algorithm 2, for each incoming packet, we need to increment $l = 20$ counters per sketch and we need to do this for two sub-sketches. We use single-precision real number counters (4 bytes each) to minimize memory I/O (space not an issue), as 7 decimal digits of precision is accurate enough for our computations. This involves 320 bytes of memory reads or writes, since each counter increment involves a memory read and a memory write. We will show next that generating realizations of p-stable distributions from two precomputed tables in order to compute sh will involve another 320 bytes of memory reads. In total, each incoming packet triggers 640 bytes of memory reads/writes. However, we will show that if implemented using commodity RDRAM DIMM 6400, our sketch can deliver a throughput of 10 million packets per second.

Our sketches can be implemented using RDRAM DIMM 6400 (named after its 6400 MB/s sustained throughput for burst accesses), except for the elephant detection module, which is to be implemented using a small amount of SRAM in the same way as suggested in [10]. RDRAM can deliver a very high throughput for read/write in *burst mode* (a series of accesses to consecutive memory locations). Since our 640 bytes of memory accesses triggered by each incoming packet consist of 4 large contiguous blocks of 160 bytes each, we can fully take advantage of the 6400 MB/s throughput provided by RDRAM DIMM 6400. Implementing our sketch using DRAM, we never need to worry about memory space/consumption, as even if we need millions of buckets in the future (we use tens of thousands right now), we are consuming only hundreds of MB; In comparison, the retail price of a commodity 2 GB RDRAM DIMM 6400 module is about $300.

Next we describe how to implement the "magic" stable hash functions sh_1, ..., sh_l used in Algorithm 2. The standard methodology for generating random variables with stable distributions $S(p)$ is through the following simulation formula:

$$X = \left[\frac{\sin(p\theta)}{\cos^{1/p}\theta}\left(\cos(\theta(1-p))\right)^{1/p-1}\right]\left[\left(\frac{1}{-\ln r}\right)^{1/p-1}\right], \tag{9}$$

where θ is chosen uniformly in $[-\pi/2, \pi/2]$ and r is chosen uniformly in $[0, 1]$ [6].

One possible way to implement these stable hash functions sh_j, $j = 1$, ..., l, is as follows. To implement each sh_j we fix two uniform hash functions uh_{j1} and uh_{j2} that map a flow identifier $pkt.id$ to a θ value uniformly distributed in $[-\pi/2, \pi/2]$, and an r value uniformly distributed in $[0, 1]$ respectively. We then plug these two values into the above

[4] In fact, even to state rigorously the theorem we would like to prove requires more space than we have here.

formula. However, computing Formula (9) requires thousands of CPU cycles, and it is not possible to perform 40 such computations for each incoming packet.

Our solution for speeding up the computation of these stable hash functions (i.e., $sh'_j s$) is to perform memory lookups into precomputed tables (also in RDRAM DIMM 6400) as follows. Note that in the RHS of (9), the term in the first bracket is a function of only θ and the term in the second bracket is a function of only r. For implementing each sh_j we now fix two uniform hash functions uh_{j1} and uh_{j2} that map a flow identifier $pkt.id$ to two index values uniformly distributed in $[1..N_1]$ and $[1..N_2]$ respectively. We allocate two lookup tables T_1 and T_2 that contain N_1 and N_2 entries respectively, and each table entry (for both T_1 and T_2) contains $l = 20$ blocks of 4 bytes each. Then we precompute $N_1 * 20$ i.i.d. random variables distributed as the term in the first bracket and fill them into T_1, and precompute $N_2 * 20$ i.i.d. random variables distributed as the term in the second bracket and fill them into T_2. For each incoming packet, we simply return $l = 20$ random values $T_1[uh_{j1}(pkt.id)][j] * T_2[uh_{j2}(pkt.id)][j]$, $j = 1, ..., 20$, as the computation result for $sh_1(pkt.id)$, $sh_2(pkt.id)$, ..., $sh_l(pkt.id)$. Since each sub-sketch requires two tables, we need a total of four tables. In our implementation, both N_1 and N_2 are set to fairly large values like 1M. Our simulation shows that stable distribution values generated this way is indistinguishable from real stable distribution values. Note that our implementation is very fast: two memory reads (4 bytes each) and a floating point multiplication for computing each $sh_j(pkt.id)$. Note also that index values $uh_{j1}(pkt.id)$ and $uh_{j2}(pkt.id)$ generated for estimating the $L_{1+\alpha}$ norm can be reused for the lookup operations performed in estimating the $L_{1-\alpha}$ norm, since all entries in these four tables are mutually independent.

For our distributed algorithm in Section 5 to work, we need all the nodes to use identical lookup tables, and identical uniform hash functions that are used to map into the lookup tables. One way to ensure the identical lookup tables is to distribute a random value to every ingress and egress node and to use it as the seed to each of their (identical) pseudorandom number generators.

8. EVALUATION

In this section, we evaluate the performance of our algorithm using real packet traces obtained from a tier-1 ISP.

8.1 Data Gathering

We deployed a packet monitor on a 1 Gbit/second ingress link from a data center into a large tier-1 ISP backbone network. The data center hosts tens of thousands of Web sites as well as servers providing a wide range of services such as multimedia, mail, etc. The link monitored is one of multiple links connecting this data center to the Internet. All the traffic carried by this link enters the ISP backbone network at the same ingress router. For each observed packet, the monitor captured its IP header fields as well as UDP/TCP and ICMP information required for the flow definition we considered.

We collected a number of five-minute traces and a one-hour trace in April 2007. We use the routing table dumped at the ingress router to determine the egress router for each packet, thus determining the OD flows to each possible egress router. Because we don't have the packet monitoring capability at egress routers, we chose to generate synthetic traffic traces at egress routers so that they contain corresponding OD flows observed at the ingress router. We can further dictate the flow size distribution at egress routers. In most cases, we make them match the flow size distribution of the ingress trace.

In the rest of the paper, we will mostly use the following two traces to illustrate our results.[5]

- *Trace 1*: A one hour trace collected at 9:41pm on April 25, 2007. It contains 0.4 billion packets which belong to 1.8 million flows. We chose one egress router so that the traffic between the origin and destination comprised of 5% of the total traffic arriving at the ingress router.
- *Trace 2*: A five minute trace collected at 10:06pm on April 25, 2007. Similar to Trace 1, the traffic between the origin and our chosen egress router comprised of 5% of the total traffic arriving at the ingress router. The traffic in this trace is purposely chosen as being a subset of the traffic for Trace 1 so that we can directly compare the performance of our algorithm for five minutes and one hour intervals.

8.2 Experimental Setup

For each trace, we repeat each experiment 1000 times with independently generated sketch data structures and compute the cumulative density function of the relative error. Unless stated otherwise, the parameters we used for each experiment were: number of buckets $k = 50,000$, number of registers in each bucket $l = 20$, sample and hold sampling rate $P = 0.001$, and one million entries in each lookup table.

In our experiments, we also define an elephant flow (for which the contribution to the entropy are computed separately) to be any flow with at least $N = 1000$ packets. We use $\alpha = 0.05$, which satisfies the constraint $\alpha < 1/\ln N$. Hence, at every ingress and egress point we have a pair of sketches computing the $L_{1.05}$ and $L_{0.95}$ norms of the traffic.

8.3 Experimental Results

Formulae 1 and 2: Recall from Section 5 that we had two formulae for estimating the L_p norms, i.e.,

- *Formula 1*: $\left(\frac{L(\vec{O})^p + L(\vec{D})^p - L(\vec{O}-\vec{D})^p}{2}\right)^{1/p}$
- *Formula 2*: $\left(\frac{L(\vec{O}+\vec{D})^p - L(\vec{O}-\vec{D})^p}{2^p}\right)^{1/p}$.

We first compare the experimental results for these two formulae. The cumulative density plots for the error of our algorithm using these two formulae for Trace 1 are given in Figure 2. We observe that both formulae have reasonably small and comparable error values. This observation also holds on all five minute traces and hence we fix and use Formula 1 for the rest of our evaluation.

Varying number of buckets: We study the effect of varying the number of buckets on the performance of our algorithm. Keeping all other parameters fixed at reasonable values, we varied the number of buckets between $k = 5000$ and $k = 80000$, with increasing factors of two. Figure 3 shows the results of Trace 2. We observe that increasing the number of buckets increases the accuracy (as expected), but with diminishing returns.

[5] The results on other five minute traces are very similar. We omit them here for sake of brevity.

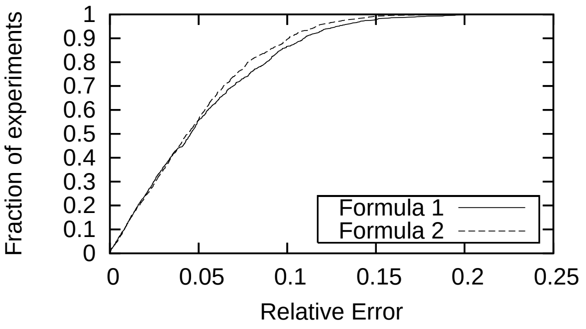

Figure 2: Comparing Formulae 1 and 2 (Trace 1)

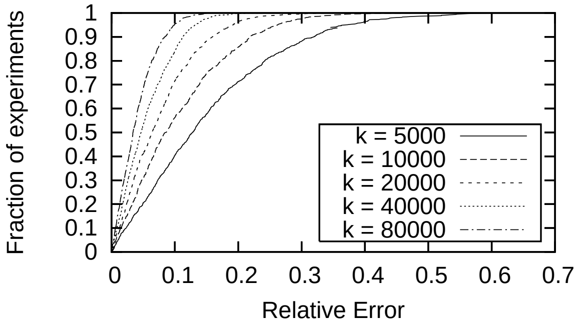

Figure 3: Varying numbers of buckets (Trace 2)

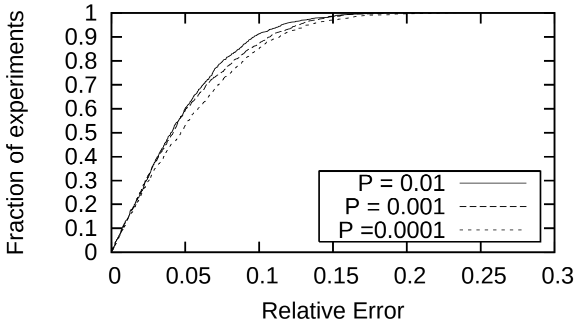

Figure 4: Varying sampling rates (Trace 2)

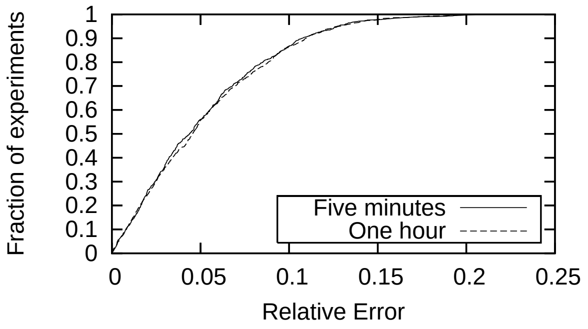

Figure 5: Comparing Traces 1 and 2

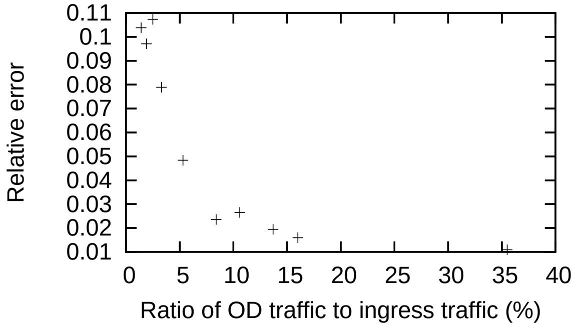

Figure 6: Varying fraction of traffic from ingress

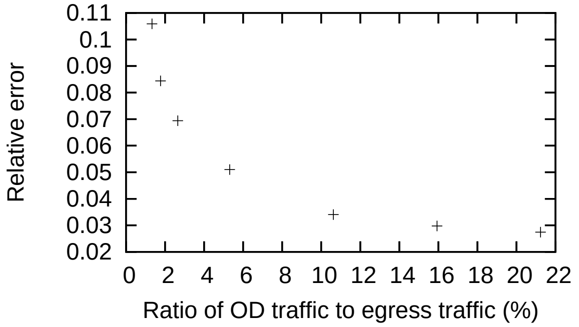

Figure 7: Varying fraction of traffic from egress

Varying sampling rate: Recall from Section 4 that we separate the computation for the large (elephant) flows by means of Sample and Hold. By varying the sampling rate we can increase or decrease the probability with which we will sample flows that are above our elephant threshold (i.e., flows of size greater than 1000). We found that the sampling rate did not affect the performance of our algorithm significantly. Figure 4 shows that, even with a small sampling rate (e.g., 1 in 1000), the elephant detection mechanism allows good overall performance for our algorithm.

Varying trace length: Figure 5 compares the cumulative density plots of the error for the five minute, Trace 2, and the one hour, Trace 1, which have the same origin and destination nodes and similar traffic distributions. We observe that even though there is an order of magnitude difference in the size of these traces, not only does the error remain comparable, but also the distribution of the error. Our experiments on different trace sizes show that the algorithm is robust to changes in the size of the trace.

Varying cross traffic: We study the variation of the accuracy of our algorithm based on what fraction of the total flow (from the source) that particular OD flow comprises. For OD flows that are very small in comparison to the volumes of traffic at the origin (and destination) we expect the performance of our algorithm to degrade since the variation of the cross traffic will begin to dominate the error of our estimator. This is demonstrated in Figures 6 and 7 for various fractions of the ingress and egress traffic (using the average of 100 runs), respectively. The complete c.d.f. for the ingress traffic is given for reference in Figure 8.

Computing Actual Entropy: We evaluate the performance of our algorithm in computing the actual entropy (as opposed to the entropy norm) of the OD flows. This computation has additional error since we need to make use of our sketch to estimate the total volume of traffic between the source and destination. In Figure 9 we observe that the er-

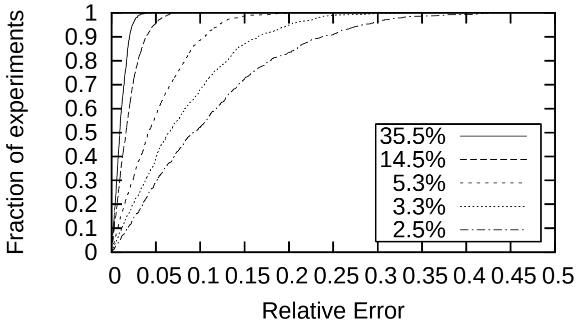

Figure 8: Varying fraction of traffic from ingress

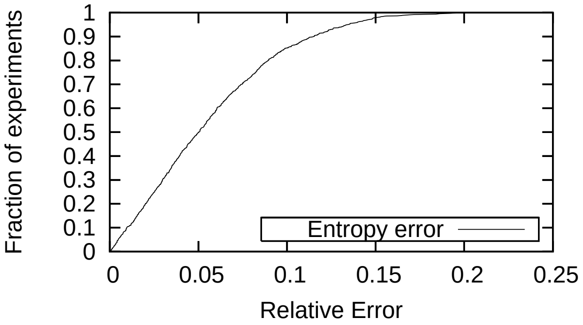

Figure 9: Error distribution for actual entropy

ror plot for the entropy is comparable to that for the entropy norm. Hence, this confirms the fact that our algorithm is a robust estimator of the entropy of OD flows.

9. RELATED WORK

There has been considerable previous work in computing the traffic matrix in a network [18, 21, 22, 23, 27]. The traffic matrix is simply the matrix defined by the packet (byte) counts between each pair of ingress and egress nodes in the network over some measurement interval. For fine-grained network measurement we are sometimes interested in the flow matrix [28], which quantifies the volume of the traffic between individual OD flows. In this paper we propose the computation of the entropy of every OD flows, which gives us more information than a simple traffic matrix and has considerably less overhead than maintaining the entire flow matrix.

In the last few years, entropy has been suggested as a useful measure for different network-monitoring applications. Lakhina et al. [15] use the entropy measure to perform anomaly detection and network diagnosis. Information measures such as entropy have been suggested for tracking malicious network activity [11, 24]. Entropy has been used by Xu et al. [25] to infer patterns of interesting activity by using it to cluster traffic. For detecting specific types of attacks, researchers have suggested the use of entropy of different traffic features for worm [24] and DDoS detection [11]. Recently, it has been shown that entropy-based techniques for anomaly detection are much more resistant to the effects of sampling [3] than other, volume-based methods.

The use of stable distributions to produce a sketch was first proposed in [12] to measure the L_1 distance between two streams. This result was generalized to the L_p distance for all $0 < p \leq 2$ in [8, 13]. This sketch data structure is the starting point for the one that we propose in this paper. The main difference is that we have to make several key modifications to make it work in practice. In particular, we have to ensure that the number of updates per packet is small enough to feasibly perform in realtime.

Other than for $p = 1, 2$, there is no known closed form for the p-stable distribution. To independently draw values from an arbitrary p-stable distribution, we make use of the formula proposed by [6]. Since this formula is expensive to compute, we create a lookup table to hold several precomputed values.

In [7] it is suggested that the stable distribution sketch can be used as a building block to compute empirical entropy, but no methods for doing this are suggested. More importantly, there are already known to be algorithms that work well for the single stream case [16] and we believe that it is for this distributed (i.e., traffic matrix) setting that the stable sketch really shines.

10. CONCLUSION

In this paper we motivate the problem of estimating the entropy between origin destination pairs in a network and present an algorithm for solving this problem. Along the way, we present a completely novel scheme for estimating entropy, introduce applications for non-standard L_p norms, present an extension of Indyk's algorithm, and show how it can be used in our distributed setting. Via simulation on real-world data, collected at a tier-1 ISP, we are able to demonstrate that our algorithm is practically viable.

11. REFERENCES

[1] A. Chakrabarti, K. Do Ba, and S. Muthukrishnan. Estimating entropy and entropy norm on data streams. In *STACS*, 2006.

[2] L. Bhuvanagiri and S. Ganguly. Estimating entropy over data streams. In *ESA*, 2006.

[3] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina. Impact of packet sampling on anomaly detection metrics. In *IMC*, 2006.

[4] G. Casella and R. L. Berger. *Statistical Inference*. Duxbury, 2nd edition, 2002.

[5] A. Chakrabarti and G. Cormode. A near-optimal algorithm for computing the entropy of a stream. In *SODA*, 2007.

[6] J. M. Chambers, C. L. Mallows, and B. W. Stuck. A method for simulating stable random variables. *Journal of the American Statistical Association*, 71(354), 1976.

[7] G. Cormode. Stable distributions for stream computations: It's as easy as 0,1,2. In *Workshop on Management and Processing of Data Streams*, 2003.

[8] G. Cormode, P. Indyk, N. Koudas, and S. Muthukrishnan. Fast mining of massive tabular data via approximate distance computations. In *ICDE*, 2002.

[9] M. Durand and P. Flajolet. Loglog counting of large cardinalities. In *ESA*, 2003.

[10] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting. In *SIGCOMM*, Aug. 2002.

[11] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred. Statistical approaches to DDoS attack detection and response. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, 2003.

[12] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *FOCS*, 2000.

[13] P. Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

[14] A. Kuzmanovic and E. W. Knightly. Low-rate tcp targeted denial of service attacks (the shrew vs. the mice and elephants). In *SIGCOMM*, 2003.

[15] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *SIGCOMM*, 2005.

[16] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang. Data streaming algorithms for estimating entropy of network traffic. In *SIGMETRICS*, 2006.

[17] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.

[18] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation:existing techniques and new directions. In *SIGCOMM*, Aug. 2002.

[19] S. Muthukrishnan. Data streams: algorithms and applications. available at http://athos.rutgers.edu/∼muthu/.

[20] J. Nolan. STABLE program. online at http://academic2.american.edu/∼jpnolan/stable/stable.html.

[21] A. Soule, A. Nucci, R. Cruz, E. Leonardi, and N. Taft. How to identify and estimate the largest traffic matrix elements in a dynamic environment. In *SIGMETRICS*, June 2004.

[22] C. Tebaldi and M. West. Bayesian inference on network traffic using link count data. *Journal of American Statistics Association*, pages 557–576, 1998.

[23] Y. Vardi. Internet tomography: estimating source-destination traffic intensities from link data. *Journal of American Statistics Association*, pages 365–377, 1996.

[24] A. Wagner and B. Plattner. Entropy Based Worm and Anomaly Detection in Fast IP Networks. In *Proceedings of IEEE International Workshop on Enabling Technologies, Infrastructures for Collaborative Enterprises*, 2005.

[25] K. Xu, Z.-L. Zhang, and S. Bhattacharya. Profiling internet backbone traffic: Behavior models and applications. In *SIGCOMM*, 2005.

[26] Y. Zhang, Z. M. Mao, and J. Wang. Low-rate tcp-targeted dos attack disrupts internet routing. In *Proc. 14th Annual Network & Distributed System Security Symposium*, 2007.

[27] Q. Zhao, Z. Ge, J. Wang, and J. Xu. Robust traffic matrix estimation with imperfect information: making use of multiple data sources. In *SIGMETRICS*, 2006.

[28] Q. Zhao, A. Kumar, J. Wang, and J. Xu. Data streaming algorithms for accurate and efficient measurement of traffic and flow matrices. In *SIGMETRICS*, June 2005.

[29] V. M. Zolotarev. *One-Dimensional Stable Distributions*, volume 65 of *Translations of Mathematical Monographs*. American Mathematical Society, Providence, RI, 1986.

APPENDIX

A. PROOFS

A.1 Proof of Theorem 2

LEMMA 9. *Let $f = f_{S^+(p)}$, $m = DMed_p$. Then estimator (1) is asymptotically normal with mean $||S||_p$ and standard deviation $\frac{1}{2mf(m)\sqrt{l}}||S||_p$.*

PROOF. Let $X_i = Y_i/||S||_p$, then X_i are i.i.d. samples from distribution $S(p)$, so $|X_i|$ are i.i.d. samples from distribution $S^+(p)$. Using the asymptotic normality of the median stated in [4, p. 483], $\sqrt{l}(\text{median}(|X_1|, \ldots, |X_l|) - m)$ is asymptotically normal with mean 0 and standard deviation $\frac{1}{2f(m)}$. Dividing by $m\sqrt{l}$ and multiplying by $||S||_p$ give us that, $||S||_p \text{median}(|X_1|, \ldots, |X_l|)/m - ||S||_p$
$= \text{median}(|Y_1|, \ldots, |Y_l|)/m - ||S||_p = \Lambda(\vec{Y}) - ||S||_p$, is asymptotically normal with mean 0 and standard deviation $\frac{1}{2mf(m)\sqrt{l}}||S||_p$. □

PROOF OF THEOREM 2. From Lemma 9,
$Pr[|\Lambda(\vec{Y})/||S||_p - 1| < \epsilon] \approx Pr[|\frac{1}{2mf(m)\sqrt{l}}Z| < \epsilon]$
$= Pr[|\frac{\epsilon}{z_{\delta/2}}Z| < \epsilon] = Pr[|Z| < z_{\delta/2}] = 1 - \delta$. □

A.2 Proofs of Propositions 4 and 5

LEMMA 10. *$x^\alpha/\ln x$ is a decreasing function on (1, N] if $\alpha < 1/\ln N$. In fact, if $\alpha < 0.085$, then $x^\alpha/\ln x < 1$ on [3,N].*

PROOF. The derivative is negative on (1,N]. $3^\alpha < \ln 3$ for $\alpha < 0.085$. □

LEMMA 11. *If a approximates b within relative error bound ϵ, then $a^{1+\alpha}$ approximates $b^{1+\alpha}$ roughly within relative error bound ϵ for small α and ϵ. Similarly for $1 - \alpha$.*

PROOF. $1-\epsilon < a/b < 1+\epsilon$, so $(1-\epsilon)^{1+\alpha} < a^{1+\alpha}/b^{1+\alpha} < (1+\epsilon)^{1+\alpha}$. Using Taylor expansion, $(1+\epsilon)^{1+\alpha} = 1+\epsilon+\alpha\epsilon+O(\epsilon^2) \approx 1+\epsilon$ for small α and ϵ. Similarly $(1-\epsilon)^{1+\alpha} \approx 1-\epsilon$. Same for $1 - \alpha$. □

PROOF OF PROPOSITION 4. We know $|c(a_i^{1+\alpha} - a_i^{1-\alpha}) - a_i \ln a_i| \le \epsilon_0 a_i \ln a_i$, sum over i gives $|c(||S||_{1+\alpha}^{1+\alpha} - ||S||_{1-\alpha}^{1-\alpha}) - ||S||_H| \le \epsilon_0 ||S||_H$. So $\lambda_0 \le 1$.

From Lemma 10 , $a_i^{1+\alpha} < a_i \ln a_i$ except for a few small numbers. The big numbers should dominate the small numbers, so we argue that for a typical flow distribution, $\sum a_i^{1+\alpha} < \sum a_i \ln a_i$, i.e. $||S||_{1+\alpha}^{1+\alpha} < ||S||_H$. Let $\lambda = ||S||_{1+\alpha}^{1+\alpha}/||S||_H$, so $\lambda < 1$. Also $||S||_{1-\alpha}^{1-\alpha} < ||S||_{1+\alpha}^{1+\alpha} = \lambda ||S||_H$.

For simplicity of notation, from now on we will use y to denote $\Lambda(\vec{Y})$ and z to denote $\Lambda(\vec{Z})$.

We know $|y - ||S||_{1+\alpha}| < \epsilon ||S||_{1+\alpha}$ with probability $1 - \delta$, so from Lemma 11, roughly $|y^{1+\alpha} - ||S||_{1+\alpha}^{1+\alpha}| < \epsilon ||S||_{1+\alpha}^{1+\alpha} = \lambda\epsilon ||a||_H$ with probability $1 - \delta$. Similarly $|z^{1-\alpha} - ||S||_{1-\alpha}^{1-\alpha}| < \lambda\epsilon ||a||_H$ with probability $1 - \delta$. So both inequalities will be true with probability at least $1 - 2\delta$. When that is the case,

$$\begin{aligned}
&|c(y^{1+\alpha} - z^{1-\alpha}) - ||S||_H| \\
&< |c(y^{1+\alpha} - z^{1-\alpha}) - c(||S||_{1+\alpha}^{1+\alpha} - ||S||_{1-\alpha}^{1-\alpha})| \\
&+ |c(||S||_{1+\alpha}^{1+\alpha} - ||S||_{1-\alpha}^{1-\alpha}) - ||S||_H| \\
&< c|y^{1+\alpha} - ||S||_{1+\alpha}^{1+\alpha}| + c|z^{1-\alpha} - ||S||_{1-\alpha}^{1-\alpha}| \\
&+ |c(||S||_{1+\alpha}^{1+\alpha} - ||S||_{1-\alpha}^{1-\alpha}) - ||S||_H| \\
&< (2c\lambda\epsilon + \lambda_0\epsilon_0)||S||_H. \quad \square
\end{aligned}$$

PROOF OF PROPOSITION 5. From Lemma 9, the error in using $\Lambda(\vec{Y})$ to estimate $||S||_{1+\alpha}$ is roughly Gaussian with mean 0 and standard deviation $(\pi/2\sqrt{l})||S||_{1+\alpha}$ (here we used value of $mf(m)$ at $p = 1$). We assume the error in using $\Lambda(\vec{Y})^{1+\alpha}$ to estimate $||S||_{1+\alpha}^{1+\alpha}$ is still roughly Gaussian with mean 0 and standard deviation $(\pi/2\sqrt{l})||S||_{1+\alpha}^{1+\alpha}$. (This is in the same spirit as $(1+\epsilon)^{1+\alpha} \approx 1+\epsilon$.) Similarly we assume the error in using $\Lambda(\vec{Z})^{1-\alpha}$ to estimate $||S||_{1-\alpha}^{1-\alpha}$ is roughly Gaussian with mean 0 and standard deviation $(\pi/2\sqrt{l})||S||_{1-\alpha}^{1-\alpha}$, which we will enlarge to $(\pi/2\sqrt{l})||S||_{1+\alpha}^{1+\alpha}$. Adding the two Gaussians gives $\sqrt{2}$ times the original Gaussian. Therefore the error is multiplied by $\sqrt{2}$ under the same probability $1 - \delta$. □