Scalable heavy-hitter identification

Rade Stanojević*

ABSTRACT

In this paper we address the problem of real-time identification of rate-heavy-hitters on very high speed links, using small amounts of SRAM memory. Assuming a nonuniform distribution of flow rates, runs (where two randomly selected bytes belong to the same flow) occur predominantly in the heaviest flows. Based on this observation we present RHE (Realtime Heavy-hitter Estimator), a measurement tool which uses a small amount of memory for extracting the heaviest flows. Using this tool a queue management algorithm called HH (Heavy-hitter Hold) is presented that approximates fair bandwidth allocations. RHE posses very low memory requirements, scales with line speeds of several tens of Gbps and covers a wide range of flow rate distributions. Measurements over real Internet traces show the high efficiency of RHE, achieving a high accuracy with a very small amount of SRAM memory. Compared to Estan-Varghese's Multistage Filters and Lucent's CATE estimator, RHE achieves up to 10 times smaller errors in measuring high-rate flows on a number of synthetic traces. Packet level ns2 simulations in a synthetic heavy-tailed environment are presented to illustrate efficacy of HH.

1. INTRODUCTION

Measurement of traffic on high speed links is essential for appropriate network management. In order to optimize performance, network operators need traffic measurements that provide insights at both quantitative and qualitative level. Long-term measurements can be used for traffic engineering in developing network architecture or rerouting traffic. Short-time measurements that monitor traffic on a scale of few seconds to few hours are needed for intrusion detection. Real time measurements can be used for managing flow control and providing QoS. Also, practical knowledge of various aspects of network traffic provides necessary feedback to researchers that work on analysis of IP networks.

*rade.stanojevic@nuim.ie.

Having its importance in mind, traffic measurement as research area has attracted significant attention in last few years. The basic problem behind this research lies in the inability of large DRAM memory devices to process packets quickly enough and fast SRAM memories to allow enough space for full per-flow state. Namely, number of concurrent flows on high speed links ($\geq 1Gbps$) can be more than a million. A naive approach, of keeping per flow state for each flow cannot be implemented on large DRAM since it is not possible to manage per-flow counters at speeds of current backbone links; moreover one cannot expect in the near future that the speeds of DRAMs (which increase approximately 7% per year) will reach speeds of backbone links (which roughly double every year)[7, 1]. On the other hand, using fast SRAM is not feasible since the size of available SRAM chips is significantly smaller than needed for keeping per-flow state for all flows at very high speeds[1, 30, 35].

Based on the observation that on most internet links small number of flows account for large amount of bandwidth the authors of the seminal paper [7] have developed 2 efficient algorithms for measuring of these heavy-hitters – flows that account for most of bandwidth. However, the algorithms are passive in sense that they measure size of flows rather than their *rates*. Efficient, real-time, measurements of flow rates rather than flow sizes would be essential in QoS per-flow management and for enforcing fairness in networks in which users use heterogenous congestion control algorithms, or do not react on congestion signals at all. In environments with flows that have a fixed sending rate, measuring the flow size and the flow rate over an interval is equivalent. In environments with TCP-like flows, where rate may increase and decrease one should be much more careful when measuring flow rates.

In this paper we develop a novel technique, called RHE, for real-time measuring of rates of flows with the highest rates on very high speed links. Then we apply this technique and describe an Active Queue Management(AQM) scheme called HH that controls fairness using information on measured rates.

For the purpose of measuring flows with highest rates we introduce a data structure that is built of memory cells. Each memory cell corresponds to certain set of flows that are mapped to it. Assuming nonuniform distribution of flow rates, information kept in a memory cell is enough to extract (with high probability) the flow with the highest rate that maps to that cell.

Measurements over synthetic streams show that RHE achieves up to 10 times smaller errors in measuring heavy flows,

compared to state of the art methods: Multistage Filters (MSF)[7] and Coincidence bAsed Rate Estimator[1] (CATE)[14]. On the other hand, number of measurements performed on the real packet traces show high accuracy of RHE with extremely small amount of memory.

The paper is structured as follows. In Section 2 we describe previous work and define the problem of interest. Section 3 will present the technique RHE for dynamic tracking of flows with highest rates. Analysis of RHE will be given in Section 4. In Section 5 we apply RHE to introduce an AQM scheme HH that will control fairness[2]. We will experimentally evaluate both RHE and HH in Section 6 and conclude in Section 7.

2. PRELIMINARIES

2.1 Previous work

The most widely used tool for traffic measurements is Cisco NetFlow[25]. NetFlow keeps per-flow state for each user in a large DRAM memory. At very high speeds NetFlow deals with the following two issues: (a) packet interarrival times can be significantly smaller than time needed for processing of a packet; (b) the amount of traffic measurement data can be enormous. To overcome these problems NetFlow uses sampling, with a sampling rate as a parameter that is set manually. However, it is clear that in order to optimize memory usage and accuracy traffic conditions must be considered when choosing the sampling rate. The paper [9] deals with this problem by introducing self-tuning NetFlow, where the sampling rate adapts to current traffic measurements. See [15] or [5] for review of sampling methods in traffic measurements.

In order to move from slow[3] DRAM to fast SRAM the authors of [7] suggest a method called Multistage Filtering (MSF) that attempts to monitor only flows with volume greater than a certain threshold, T, in a certain measurement interval and to neglect all smaller flows. In order to meet this goal, MSF contains two components: hash filters and a flow container. On every packet arrival the flow identifier is used as an index for computing hash in several stages. Each stage uses an independent hash function and each counter at every stage contains the aggregate amount of traffic for all flows that map to that counter. The volume of a flow is then estimated as the least value of all counters that this flow maps to. If the estimated volume of a flow is greater than the threshold T the flow is added to the flow container. It is obvious that if a certain flow has volume greater than T then the estimated volume will be at least T and it will be added to the flow container, assuming that there is space for it. In order to prevent flows with volume smaller than T being wrongly identified as large, two enhancements are introduced: shielding and conservative updates. Assuming that the sending rates do not vary over the measurement intervals and that the objective is finding the total volume over the measurement interval rather than the sending rate of a flow, MSF together with conservative updates and shielding achieve this goal with high accuracy for heavy hitters in environments with heavy-tailed flow size distributions. For flow control one needs rate rather than volume of a flow over long measurement intervals and a quick response to this information. The main contribution of this paper is RHE which will meet these requirements using only small and fast SRAM memory.

In [19] in order to save memory the authors propose a mechanism, called RATE, for estimating elephant flows by counting *runs*: events that correspond to the situation where an arriving packet belongs to the same flow as some sampled packet that has recently arrived at the link. However, to give an accurate estimate of elephant flows on links with large number of users RATE needs a long time. In order to solve this scalability problem, the same authors propose ACCEL-RATE[13] and CATE[14] which require more per packet processing and memory. As it is shown in [14] these algorithms "*...tend to slightly over-estimate flow rates.*" over real IP traces. This problem is caused by two factors: (a) the bursty nature of internet traffic and (b) the fact that many flows do not have a constant rate but rather a variable rate, or even more drastically on/off patterns. Factor (a) can be resolved by introducing buffer for counting runs. Factor (b) appears to be much harder to resolve and is the main cause of inaccuracy of RATE-like algorithms that try to use a run counter as the main source of measurements. However, counting runs can give some important insights on flow rates. We are going to exploit this idea, but in a different way, as we will see in later sections.

Another interesting approach in traffic measurements is the accurate estimation of the number of very small flows as they would indicate potential DDoS or Internet worm attacks. In [22] the authors use a Bayesian statistical method, Expectation Maximization, together with the efficient implementation of sequence of counters proposed in [30] to give a very accurate estimation of the flow size distribution[4]. A potential weakness of this work is a vulnerability in the number of counters. Namely it is shown in [22] that accuracy of the proposed algorithm drops considerably when number of flows is larger than 2*(number of available counters). See [**?**] for another memory efficient scheme that utilizes a multiresolution-Bloom-filter technique to capture the (heavy) flow rates.

For caching techniques used in heavy-hitter identification see [**?**] and references therein.

Fair queueing has been an important research area in networking for the previous two decades, motivated by idea that in heterogenous network conditions the service that a user gets should not depend on the aggressiveness of its congestion control algorithm. Various approaches include fair scheduling [3, 31], keeping state information on overly aggressive users [23, 24, 28] or change of IP infrastructure and exploiting state information written in IP header for controlling fairness as in [33, 17]. The way information is extracted on highly aggressive flows in [23, 24, 28] is by naive sampling and requires relatively large amounts of memory on very high speed links. HH goes further in this direction and by exploiting efficacy of RHE we aim to control flow rates even on very high speed links with low memory usage.

[1]CATE is the final version of runs based estimators developed by Lucent group and its performance is strictly better than the previous incarnations RATE[19] and ACCEL-RATE[13].

[2]Throughout this paper we accept max-min definition of fairness[32].

[3]Current access times are in range $50 - 100ns$ for DRAM, and $2 - 6ns$ for SRAM[35].

[4]Note that problem of estimation of flow size/rate distribution is different from the problem of heavy-hitter identification since the first problem ignores identities of the flows.

2.2 Problem definition

A flow can be defined as set of packets that satisfy a certain condition that is called the flow key. In most cases the flow key is defined as some function of the following information kept from a packet header: IP address of source and destination, port numbers, protocol ID, etc.

Our goal is to do real-time accurate measurement of the rates of flows with the highest rates with the finite amount of SRAM memory. In order to do so, the most important question is "What flows have the biggest rate?". Thus the problem of interest for us is following:

For a given positive integer M identify and monitor as many as possible among the M flows with the highest sending rate using approximately $10 \cdot M$ bytes of SRAM memory.

Important assumption that have to be taken in consideration is that per-packet processing should be less than few hundreds of ns, or equivalently less than the time of processing an average-sized packet at given high-speed link. Thus, although we are concerned with elephant flows, our objective is to monitor the M "biggest" flows rather than flows with size greater than some threshold T.

The notion of flow rate is not strictly defined in networking literature. However, there is consensus that for applications related to flow control, flow rates should involve some kind of weighted averaging in order to smooth out possible large variations that can occurs as a consequence of bursty traffic and on/off traffic patterns. For example, authors of [33] define flow rate as a weighted average that depends on inter-arrival time T between the previous and the current packet of size l:

$$A_{new} = (1 - e^{-T/K})\frac{l}{T} + e^{-T/K} A_{old}$$

Where A_{new} and A_{old} denote estimated rates for the current and previous packet respectively and K is a smoothing parameter. Implementing a strategy like this would require keeping per-flow time-stamps as well as using a "CPU-expensive" exponential function. Since our goal is the estimation of rates with as small amount of memory usage as possible that runs at line speed for high speed links, we will try to avoid keeping the time-stamps and use functions with lower computational cost in our definition of flow rate.

We define the sending rate of flow f in the following way: we divide time from beginning of measurement into sub-measurement intervals of length δ. Let $vol^{(f)}(a, b)$ be volume of data in the interval (a, b) in bytes. At time $t \in (s\delta, (s+1)\delta]$, the rate of flow f in bytes per second is defined as:

$$r_f(t) = \frac{1}{q\delta + (1-q)(t - s\delta)}(q\delta \cdot r_f(s\delta) + (1-q) \cdot vol^{(f)}(s\delta, t)) \quad (1)$$

This can be seen as weighted averaging of rates over sub-measurement intervals with weighting parameter q: packets that arrive in the last sub-measurement interval have weight $1 - q$, packets that arrive in the previous sub-measurement interval are accounted with weight $q(1 - q)$,..., packets that arrive in the k-th previous sub-measurement interval are accounted with weight $q^k(1 - q)$. Throughout this paper we will calibrate parameters q and δ in such fashion that weights of bytes decreases approximately exponentially with factor e per second. Weight of a byte that arrived at time $t = \tau_0 - 1$ is $q^{-1/\delta}$ times[5] smaller than weight of a byte arrived at time $t = \tau_0$. Thus, our calibration rule is $q^{1/\delta} \approx e^{-1}$. For q close to 1, we can approximate:

$$e^{-1} \approx q^{1/\delta} = \left((1 - (1-q))^{\frac{1}{1-q}}\right)^{\frac{1-q}{\delta}} \approx e^{-\frac{1-q}{\delta}}.$$

Therefore, choosing q and δ to satisfy a simple rule $\delta = 1 - q$, approximately implies exponential decrease of weights with rate of e per second.

Remark. Weighted averaging (or low pass filtering) is a technique widely used for the de-noising of noisy data. While other more subtle signal-processing methods are possible (see for example [4]), in our case the weighted averaging performs good in the de-noising flow rates and can be effectively implemented.

We will now show how one can effectively compute rate of a flow f defined by (1). Let co be a counter that is incremented by the packet size of each arriving packet that belongs to flow f. At the time $t_s = s\delta$, value of the counter co is updated by $co \leftarrow q \cdot co$. Now, at time $t \in (s\delta, (s+1)\delta]$ the rate of flow f is given by

$$r_f(t) = co \cdot \frac{1-q}{q\delta + (1-q)(t - s\delta)}$$

The RHE, measurement tool described in next section, is motivated by applications in flow control and therefore is designed for measuring flow rates. However, for the purpose of measuring flow sizes one can take $\delta = 1$ and discard the scaling factor. With this minor change one can adapt RHE for flow sizes measurement but throughout this paper we assume that $q < 1$ and that RHE measures rates rather than sizes.

2.3 How to check if an element of the set is maximal?

Here we present the argument that can help to build the intuition behind RHE.

Suppose that we have a set A of positive real numbers, that sum up to 1: $\sum_{x \in A} x = 1$. How to check weather one them, say $x_1 \in A$ is the maximal element of the set A using small amount of memory? In some extreme cases no additional memory is needed to answer this question[6]. Suppose now that value of $m_d = \sum_{x \in A} x^d$ is available as well, for some $d > 1$. Then:

$$\lim_{d \to \infty} \frac{m_d}{x_1^d} = \lim_{d \to \infty} \sum_{x \in A} \left(\frac{x}{x_1}\right)^d = 1 \text{ if } x_1 = \max(A)$$

and

$$\lim_{d \to \infty} \frac{m_d}{x_1^d} = \lim_{d \to \infty} \sum_{x \in A} \left(\frac{x}{x_1}\right)^d = +\infty \text{ if } x_1 < \max(A).$$

In practice for a finite d, one can chose a threshold M such that x_1 is declared as maximal if and only if $\frac{m_d}{x_1^d} < M_d$. By choosing the appropriate M, the confidence of the above decision can be very high as we will see in next section.

In our context, of heavy-hitter identification, set A will corresponds to the set of flow rates (typically few dozens of flows) that map to a cell. By using the idea from RATE, by counting runs, one can obtain an estimate of m_2 with

[5] We assume that δ is measured in seconds; then $1/\delta$ is the number of sub-measurement intervals in one second.

[6] For example, if $x_1 > 0.5$ then it is clearly the largest element of A.

cRate (2B)	*eqPairs* (2B)
rateTyp (2B)	*SampledB* (2B)
typPkt (2B)	LF (1bit)

Figure 1: Layout of a memory cell.

very small memory overhead and small computational complexity. Using higher powers ($d > 2$) is possible as well but it would increase memory overhead needed for calculating m_d, and our initial experiments show no benefit by choosing higher d so we concentrate on $d = 2$ in the rest of the paper.

3. RHE

In this section we present tool for measuring flows with the highest rates. The basic idea is as follows: at each packet arrival a hash of a flow is computed and the packet is mapped to a memory cell. Since many flows can map to same memory cell, our goal is to identify the one with highest sending rate. In order to do it we introduce data structure sketched on Figure 1.

A memory cell contains *cRate*, information on aggregate rate of all flows that map to this cell and *rateTyp*, information on rate of a flow that is a candidate for the highest rate among flows that maps to this cell that is further identified by *typPkt*: another 16bit hash value of its flow key. Counters *cRate*, *rateTyp* and *eqPairs* (which will be described shortly) mimics the technique introduced in Section 2.2, for tracking the rate with one counter without timestamps. Thus whenever a packet from a flow that maps to the memory cell arrives, its size is added to counter *cRate*; if it belongs to flow with hash value equal to *typPkt*, counter *rateTyp* is incremented by size of arriving packet. Finally at the end of each sub-measurement interval of length δ we set

$$(cRate, rateTyp, eqPairs) \leftarrow q(cRate, rateTyp, eqPairs),$$

with $q \in (0, 1)$ as a weighted averaging parameter. By doing this we follow the definition of the flow rate given by (1), so the aggregate flow rate at time $t \in (s\delta, (s+1)\delta]$ in bytes per second is given by $cRate \cdot \frac{1-q}{q\delta+(1-q)(t-s\delta)}$ and similarly the rate of "candidate" flow is given by $rateTyp \cdot \frac{1-q}{q\delta+(1-q)(t-s\delta)}$. The question is "How do we know if the candidate is the one with highest sending rate?". To answer this question we exploit the heavy tailed nature of the Internet flow rate distribution and introduce an idea that is at the core of our method and applies the reasoning discussed in Section **??**.

Each memory cell contains *SampledB*: a 16bit hash value of flow ID for some uniformly sampled byte (rather than packet) that arrived in this cell. On each arrival *SampledB* is compared with the arriving packet's 16bit hash value and if they match then the size of arriving packet is added to counter *eqPairs* [7]. As we already said, at the end of each sub-measurement interval of length δ, the counter *eqPairs*[8] is set to $q \cdot eqPairs$. We claim that once we have this information then we can say with high confidence if the flow represented by *typPkt* has highest rate (or one of the highest rates) among flows that map to this cell. To support this statement we will have to understand meaning of the counter *eqPairs*. Let $r_1, r_2 \ldots, r_n$ be the rates of all flows that map into same cell and let R be the aggregate rate: $R = r_1 + \cdots + r_n = cRate\frac{1-q}{q\delta+(1-q)(t-s\delta)}$. Sampling *SampledB* byte-wise and assuming no hash collision we have that

$$SampledB = i \quad \text{with probability} \quad z_i = \frac{r_i}{R}.$$

Therefore expected value of *eqPairs* can be estimated by:

$$E(eqPairs) = \frac{q\delta + (1-q)(t-s\delta)}{1-q} \sum_{i=1}^{n} r_i P(SampledB = i) =$$

$$= \frac{q\delta + (1-q)(t-s\delta)}{1-q} \cdot \sum_{i=1}^{n} (R \cdot z_i) \cdot z_i = cRate \sum_{i=1}^{n} z_i^2 \quad (2)$$

In the large number of packets-per-second regime variance of *eqPairs* is small and $\sum_i^n z_i^2$ can be accurately approximated with $eqPairs/cRate$. On the other hand, for the Internet-like (heavy-tailed) flow rate distributions $\sum_i^n z_i^2$ is dominated by the maximal value of z_i.

To see this we plot histograms of $\sqrt{\sum_i^n z_i^2}/\max(z_i)$ for six different cumulative distribution of z_i given in Figure 3. This six CDF, corresponds to Pareto distribution with $k = 0.5, 1, 1.5, 2, 2.5, 3$ and $m = 40$ ($P(X_k > m + i) \sim (\frac{1}{m+i})^k$). Figure 3 contain histograms for $n = 20$; Figure 4 contain histograms for $n = 100$.

As we can see from Figures 3 and 4, with very high confidence one can declare that $\sqrt{\sum_i^n z_i^2}/\max(z_i)$ is less than 2.5 (in the first case) or 3.5 (in the second case). We use this to define a test for deciding if the candidate flow, represented by its hash value *typPkt* and with rate *rateTyp* is the "high-rate" flow. We know that $rateTyp = z_i \cdot cRate$ for some $i \in \{1, \ldots, n\}$. By using estimate (2), we conclude that if $\frac{\sqrt{\frac{eqPairs}{cRate}}}{\frac{typRate}{cRate}}$ is large, than that implies that candidate flow is not one with maximal rate and we should continue with search for maximal one. Formally, at the end of each

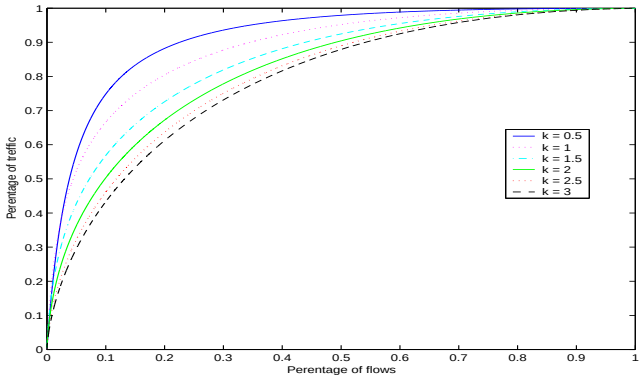

Figure 2: CDF for Pareto distribution for $k = 0.5, 1, 1.5, 2, 2.5, 3$, $m = 40$.

[7]The idea of the comparing the arriving packet's flow ID with the flow ID of some previously arrived packet has been exploited in AQM design in Stabilized RED[27] and CHOKe[29] as well as in recently proposed traffic estimators [19, 14].

[8]Following the terminology of RATE[19], counter *eqPairs* can be seen as the total number of runs weighted by packet sizes.

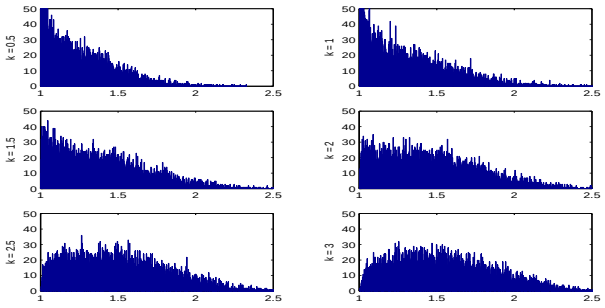

Figure 3: Histograms of $\sqrt{\sum_i^n z_i^2}/\max(z_i)$, $n = 20$.

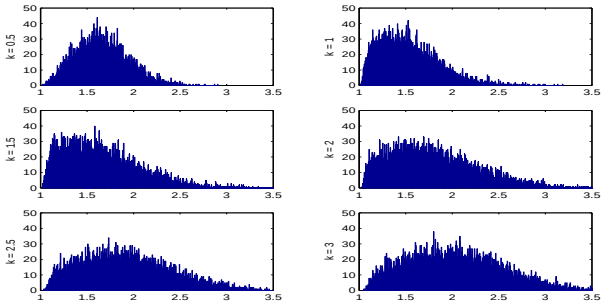

Figure 4: Histograms of $\sqrt{\sum_i^n z_i^2}/\max(z_i)$, $n = 100$.

sub-measurement interval of length δ, we do test described in Figure 5. One can notice that we use the expensive $\sqrt{\cdot}$ function in defining the test. However one can ask equivalently if $eqPairs \cdot cRate > (c_0 \cdot typRate)^2$. We emphasize again that this test is done only once in sub-measurement interval of length δ and therefore its expense is not essential.

. if $\sqrt{\frac{eqPairs}{cRate}} > c_0 \frac{typRate}{cRate}$
. REPLACE $typPkt$ WITH $SampledB$.
. end if

Figure 5: Testing of a candidate for high-rate flow.

Here $c_0 \in [1, 4]$ is a parameter that should be chosen to reflect traffic condition. In the section 4.2 we will show how to calibrate c_0.

Thus we have described an algorithm that will eventually give us the hash of the flow ID for the flow that satisfy $\sqrt{\frac{eqPairs}{cRate}} < c_0 \frac{typRate}{cRate}$. This flow might not be the largest among flows that map to the same cell, but in that case,

$$(\frac{typRate}{cRate})^2 + \max(\frac{r_i}{cRate})^2 \leq \sum_i^n z_i^2 < c_0^2 (\frac{typRate}{cRate})^2.$$

Which means that

$$typRate \geq \sqrt{\frac{1}{c_0^2 - 1}} \max(r_i).$$

Having some kind of heavy-tailed flow rate distribution, we can chose c_0 to be in the interval $(2, 4)$ and therefore the presented algorithm will give us a flow with a rate that is either the flow with largest rate or a flow whose rate is within an order of magnitude of the largest rate among flows that map to that cell.

. if $\frac{1-q}{q\delta+(1-q)(t-s\delta)} cRate \sqrt{\frac{eqPairs}{cRate} - (\frac{typRate}{cRate})^2} > T$
. $LF \leftarrow 1$
. else
. $LF \leftarrow 0$
. end if.

Figure 6: LF bit update rule

Sometimes two or more heavy flows can map to same cell. In order to identify as many large-rate flows as possible, we divide the available memory of M memory cells into several stages. The first stage with size $\lfloor M(1-\rho) \rfloor$, the second with size $\lfloor M(1-\rho)\rho \rfloor$, the third with size $\lfloor M(1-\rho)\rho^2 \rfloor$ and so on. Since we want to reduce per-packet processing we will use small ρ, say $\rho \leq 1/2$. At every packet arrival, if the packet belongs to the flow represented by $typPkt$ its rate is estimated by $rateTyp \cdot \frac{1-q}{q\delta+(1-q)(t-s\delta)}$. If the packet do not belong to the flow represented by $typPkt$ and if the bit LF (bit LF is described in the next paragraph and indicates existence of some Large Flow among non-$typPkt$ flows that map to the cell) is set to 1 the packet is hashed in one of cells in the next stage. If arriving packet does not belong to the flow represented by $typPkt$ and LF bit is 0, the flow the arriving packet belongs to is identified as small and its rate is estimated by $T/2$, where T is threshold that will be introduced in next paragraph.

In order to allow as few small flows to pass this stage as possible, the bit LF bit is updated once or several times (depending on abilities of hardware) per sub-measurement interval in the way sketched on Figure 6.

By doing this we ensure that if there is a flow that sends at a rate greater than threshold T, it will be allowed to go to next stage. Indeed, suppose that LF is set to 0 then

$$T \geq \frac{1-q}{q\delta+(1-q)(t-s\delta)} cRate \sqrt{\frac{eqPairs}{cRate} - (\frac{typRate}{cRate})^2}$$

$$= \frac{1-q}{q\delta+(1-q)(t-s\delta)} cRate \sqrt{\sum_{i=1}^n z_i^2 - (\frac{typRate}{cRate})^2} =$$

$$= \frac{1-q}{q\delta+(1-q)(t-s\delta)} cRate \sqrt{\sum_{i \neq typPkt}^n z_i^2} \geq$$

$$\geq \frac{1-q}{q\delta+(1-q)(t-s\delta)} cRate \max(z_i \mid i \neq typPkt) =$$

$$= \max(r_i \mid i \neq typPkt).$$

Thus, if there is at least one flow that maps to the cell that is not represented by $typPkt$ and with rate greater than T, then the LF bit will not be set to 0 and therefore it will be allowed to pass to next stage (together with all other non-$typPkt$ flows). Note that although LF bit set to 1 does not necessary mean that there is a flow with rate greater than T among the flows that pass from the cell to the next stage, it will indicate the existence of a flow with rate that is within an order of magnitude of T in this set of flows, assuming heavy-tailed flow rate distribution. In other words, if there does not exist a flow whose rate is within an order

. INPUT - PACKET: pkt, INTEGER: s
. $CELL \leftarrow f_s(ID(pkt))$ * HASH ON FLOW ID OF pkt
. $cRate(CELL) \leftarrow cRate(CELL) + size(pkt)$
. $h \leftarrow f_0(ID(pkt))$ * SECOND HASH
. if $(h == typPkt(CELL))$
. $rateTyp(CELL) \leftarrow rateTyp(CELL) + size(pkt)$
. end if
. if$(h == SampledB(CELL))$
. $eqPairs(CELL) \leftarrow eqPairs(CELL) + size(pkt)$
. end if
. WITH PROBABILITY $p = \frac{size(pkt)}{MaxPktSize}$ do
. $SampledB(CELL) \leftarrow h$
. end do

Figure 7: Per-packet update of memory cell at stage s.

of magnitude of T, then LF bit will be set to 0 and these flows will not be processed in the next stage.

In Subsection 3.2 we will show how to adapt the value of threshold T in order to ensure effective usage of memory. Until then we will assume that T is a parameter that is chosen manually.

3.1 Implementation issues

The data structure used in RHE is composed of M memory cells (with structure sketched at Figure 1) divided into the number of stages whose sizes decrease roughly geometrically with coefficient ρ. At the beginning of the measurement all values of all memory cells are initialized to 0. Whenever a packet arrives the hash function of its flow key is computed. This maps the packet to one memory cell and the counter *cRate* is incremented by the size of arriving packet in bytes. Another hash function is computed on the flow identifier of the arriving packet that maps each flow to a 16bit integer h. If h is equal to value *typPkt* then the counter *rateTyp* is incremented for the size of the arriving packet in bytes. Also, if h is equal to *SampledB* then the counter *eqPairs* is incremented by the size of the arriving packet. Finally with probability $p = \frac{PktSize}{MaxPktSize}$ the *SampledB* field is updated with h. Here the parameter $MaxPktSize$ is maximal possible size of a packet that can be seen at the link. Since more than 99.99% of all packets in the Internet are 9000B or less we set $MaxPktSize = 9000$[9].

Pseudo code for the processing of a packet at stage s is given in Figure 7. The complete sequence of actions for an arriving packet is shown on Figure 8

In order to avoid random number generation that might consume a lot of CPU time, we suggest the following: one register should contain 16bit integer RN. Instead of calling a random number generator to update *SampledB* we add the packet size to RN: $RN \leftarrow RN + size(pkt)$. If $RN \geq MaxPktSize$ then update *SampledB* and set $RN \leftarrow ((RN+1) \ mod \ MaxPktSize)$, otherwise do nothing. By doing this, the value RN will mimic random integer uniformly distributed on $[0, MaxPktSize-1]$ and the probability that *SampledB* is updated is proportional to the size of the arriving packet. Histograms of RN using the first million arrivals of MRA trace[26] shows uniform distribution of RN on $[0, MaxPktSize-1]$.

[9]Once MTU increases for a significant amount of traffic, then $MaxPktSize$ can be increased to follow this change.

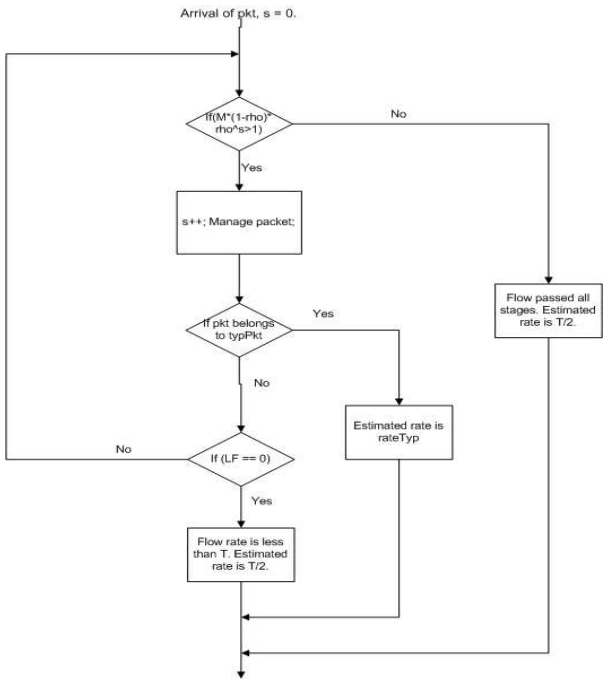

Figure 8: Flow chart

If we do not require 100% accuracy, we can reduce the number of bits of a counter *co* for k using the following argument. Whenever a new packet, with size S bytes arrives, instead of adding S to the counter *co* we can add $\lfloor S/2^k \rfloor$. Also, we can use one register for storing RN_1, a $(k+1)$bit number that would mimic a random number in same way as RN. Namely on each packet arrival update RN_1 with the sum of the last k binary digits of S and RN_1. If RN_1 is greater than 2^k, then add 1 to the counter *co* and set $RN_1 \leftarrow RN_1 \ mod \ 2^k$, otherwise do nothing. Assuming full randomness and independence of RN_1, the strong law of large numbers ensures that the difference between a counter measured in 2^k bytes and the total amount of arrived traffic is small. Taking $k = 10$ can significantly reduce memory requirements, with small (relative) errors for heavy hitters, see [**?**].

3.2 Self tuning of threshold T

The threshold T is used by RHE for reducing amount of traffic that passes from one stage of memory cells to another. Using too high value of T might cause underutilization of memory cell space in higher level stages and a loss of accuracy for flows with rate less than T, since RHE does not care about accuracy of flows with sending rate less than T. On the other hand, setting T too low results in a large amount of traffic passing from one stage to another. This will imply the following undesirable features. Firstly, since the number of cells per stage decreases geometrically, the number of flows that will map to the same cell grows implying a loss of confidence in our test (Figure 5) as well as increasing the time to search for a flow with one of the highest rates. Secondly, a large number of packets passing "threshold test" would imply larger average per-packet processing times that result in unacceptably high costs in the context of very high speed links. Knowing the flow rate distribution, one can

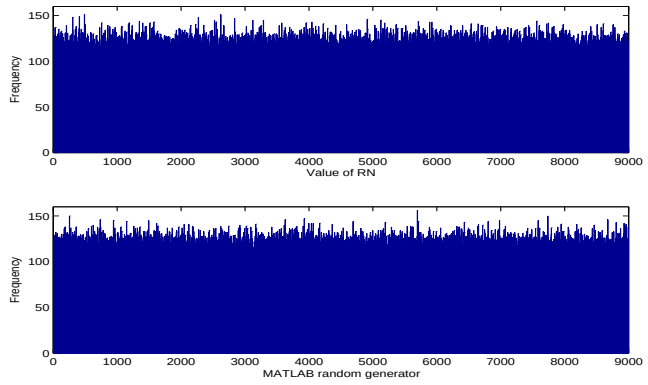

Figure 9: Histogram of RN using trace MRA(top) and appropriate histogram of random numbers generated by MATLAB random generator.

manually set a threshold that would be close to optimal for a given amount of memory cells[10].

However we propose a simple adaptation algorithm that will control amount of traffic that passes from one stage to another. The basic idea is the following: if the number of cells with the LF bit set to 1 is less then half of number of cells at the first stage, then the filtering should be amplified and so T should be increased, otherwise T should be decreased. Our algorithm uses simple MIMD[11] strategy with parameters $\alpha_1 = 1.1$, $\alpha_2 = 0.9$ that updates T once per sub-measurement interval. Pseudo code is given on Figure 9.

ONCE PER SUB-MEASUREMENT INTERVAL OF LENGTH δ
. if $((\sum LF$ *on stage 1*$)>0.5\cdot($*Size of stage 1*$))$
. $\quad T \leftarrow T \cdot 1.1$
. else
. $\quad T \leftarrow T \cdot 0.9$
. end if

Figure 10: MIMD algorithm for adaptation of threshold T.

4. CALIBRATION OF PARAMETERS

4.1 Effect of ρ

In this section we evaluate how the choice of ρ can affect the number of identified top flows. Going back to the question stated in Section 2.2, our goal was monitoring of as much as possible of the largest M flows with the memory equivalent to the M memory cells. A perfect algorithm would identify and monitor all M largest flows. However, using RHE there might exist "bad" cells that no one of the largest M flows map to. Since the total number of cells is M, the number of the largest M flows that can be monitored is at most (M - number of "bad" cells). By estimating the number of "bad" cells we will give bounds on the number of large flows that can be tracked by M memory cells. We will see that these bounds are not far from simulation results presented in Section 6.

Questions of interest are as follows:

(1) Estimate G - the number of cells that contain at least one of the largest M flows. We denote the set of the largest M flows by L_M.

(2) Estimate VG - the number of cells that contain at least one of the largest $M/2$ flows. We denote the set of the largest $M/2$ flows by $L_{M/2}$.

By following analysis we will estimate G and VG as function of ρ. Since we are interested in the upper bounds for G and VG, we will assume, for simplicity, that the threshold is such that all L_M flows passes from one stage to another. This will slightly decrease the accuracy of our bound, but main objective of this section is to give some feeling on how much we can expect from RHE.

Let a_k be the average number of flows that map to one cell at k-th stage. Denoting by N_k the total number of flows at k-th stage, we have that

$$a_k = \frac{N_k}{M\rho^{k-1}(1-\rho)}.$$

Probability that among a_1 flows at first stage there is no one from set L_M is

$$P((f_1 \notin L_M) \wedge \cdots \wedge (f_{a_1} \notin L_M)) = P(f_1 \notin L_M)^{a_1} =$$

$$= \left(1 - \frac{M}{N_1}\right)^{a_1} = \left(\left(1 - \frac{1}{a_1(1-\rho)}\right)^{a_1(1-\rho)}\right)^{\frac{1}{1-\rho}} \approx e^{-\frac{1}{1-\rho}}$$

Therefore at first stage, the number of "bad" cells one can expect is roughly:

$$B_1 = M(1-\rho)e^{-\frac{1}{1-\rho}}.$$

Thus, at first stage one can expect roughly $G_1 = M(1-\rho) - B_1$ "good" cells. In the perfect case (where each cell extract the flow with the highest rate) at the second stage one can expect approximately $M - G_1 = M(\rho + (1-\rho)(1-e^{-\frac{1}{1-\rho}}))$ flows from L_M. Therefore, the probability that a cell from the second stage is "bad" is

$$P((f_1^{(2)} \notin L_M) \wedge \cdots \wedge (f_{a_2}^{(2)} \notin L_M)) = \left(P(f_1^{(2)} \notin L_M)\right)^{a_2} =$$

$$= \left(1 - \frac{M - G_1}{N_2}\right)^{a_2} = \left(1 - \frac{\rho + (1-\rho)e^{-\frac{1}{1-\rho}}}{a_2\rho(1-\rho)}\right)^{a_2} \approx$$

$$e^{-\frac{\rho+(1-\rho)e^{-\frac{1}{1-\rho}}}{\rho(1-\rho)}}.$$

Thus the number of "bad" cells on the second stage is roughly

$$B_2 = M\rho(1-\rho)e^{-\frac{\rho+(1-\rho)e^{-\frac{1}{1-\rho}}}{\rho(1-\rho)}}.$$

Similarly, at k-th stage the number of bed cells can be approximated with

$$B_k = M\rho^{k-1}(1-\rho)e^{-\frac{\rho^{k-1}+(B_1+\cdots+B_{k-1})/M}{\rho^{k-1}(1-\rho)}}.$$

[10]For example, with M memory cells one possible approach is to chose a T value around $M/2$-th highest flow rate.

[11]Multiplicative-Increase, Multiplicative Decrease. Other control strategies are possible as well, but in our case MIMD is satisfactory.

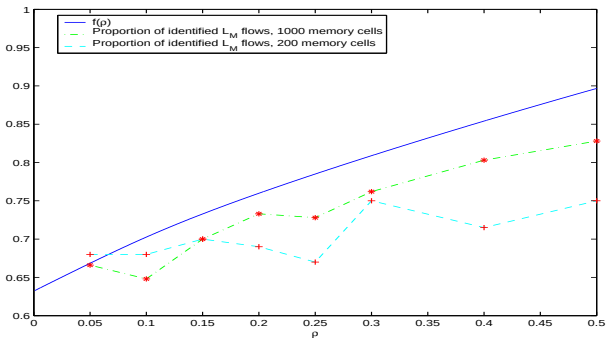

Figure 11: $f(\rho)$ - Bound on the proportion of identified flows from L_M for $\rho \in (0, 0.5)$.

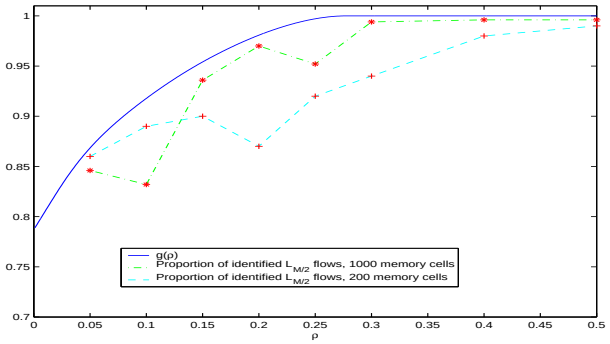

Figure 12: $g(\rho)$ - Bound on the proportion of identified flows from $L_{M/2}$ for $\rho \in (0, 0.5)$.

Having this value the proportion of identified L_M flows is upper-bounded by

$$f(\rho) = \frac{M - \sum_{k=1}^{\infty} B_k}{M}. \quad (3)$$

On Figure 10 we plot quantity $f(\rho)$, for $\rho \in (0, 0.5)$.

From Figure 10 we can see that by choosing the higher ρ we can enforce the lower number of "bad" cells. The price for this would be a larger number of stages and therefore a larger per-packet processing time.

The bound given by (3) gives a rough approximation of G - the number of "good" cells as function of ρ. Number of identified L_M flows is not larger than

$$G = f(\rho) \cdot M. \quad (4)$$

Using similar arguments we obtained bound for VG:

$$VG = g(\rho)\frac{M}{2}$$

where

$$g(\rho) = \frac{\sum_{k=0}^{\infty} A_k}{\frac{M}{2}}. \quad (5)$$

And sequence A_k satisfies: $A_0 = 0$ and

$$A_k = (1-\rho)\rho^{k-1}M(1 - e^{-\frac{\frac{1}{2} - (A_0 + \cdots + A_{k-1})/M}{\rho^{k-1}(1-\rho)}}).$$

The plots in Figures 10 and 11 also contain simulation results on a proportion of identified L_M and $L_{M/2}$ flows with RHE for various values of ρ. Two sets of simulations were performed over 10000 flows with cumulative distribution of flow rates given by Pareto distribution with $k = 0.5, m = 40$, depicted in Figure 3: one with 1000 memory cells another with 200 memory cells. Since the number of flows per memory cell in the second case is larger than in the first case, the test for identifying the highest flow has lower confidence in the second than in the first case. Therefore we see that the proportion of identified L_M and $L_{M/2}$ flows with 200 memory cells is slightly lower than with 1000 memory cells (although we see for $\rho \leq 0.1$ the opposite effect; this effect is consequence of threshold adaptation that is neglected in our analysis).

4.2 Choosing c_0

We saw in the Section 3 that if n numbers, $z_1, \ldots, z_n$, are drawn with some heavy tailed distribution $\mathcal{D}$ then the distribution $\sqrt{\sum_i^n z_i^2}/\max(z_i)$ is concentrated close to 1; see Figures 3 and 4. Parameter c_0, that defines test given on Figure 5, should be chosen is such fashion that following probability is minimized:

$$p_0(\mathcal{D}, n, c_0) = Prob((z_1, \ldots, z_n) | \frac{\sqrt{\sum_i^n z_i^2}}{\max(z_i)} > c_0)$$

Value p_0 represents the probability that test will not eventually finish its search, once the heaviest flow is found[12]. Note that even in that case (test does not finish), the maximal flow will be hashed to the next stage(s) (and will be given chance for identifying there) as long as its rate is greater than the threshold T.

Figure 12 depicts (empirically obtained) values of c_0 for which $p_0 = 0.02$ when z_i are drawn with Pareto distribution for two different values of parameter k and $m = 40$. Formally it depicts graphs of $\bar{c}_0(Pareto_1, n)$ and $\bar{c}_0(Pareto_3, n)$ for the distributions $Pareto_1$ $(k = 1, m = 40)$ and $Pareto_3$ $(k = 3, m = 40)$, where $\bar{c}_0$ is defined by:

$$\bar{c}_0(\mathcal{D}, n) = x_0 \quad \text{for which} \quad p_0(\mathcal{D}, n, x_0) = 0.02$$

Throughout this paper we assume that for the distribution of flow rates $\mathcal{D}$ holds following

Assumption 1.

$$\bar{c}_0(\mathcal{D}, n) \leq \bar{c}_0(Pareto_3, n) \quad (6)$$

Informally, it says that distribution $\mathcal{D}$ is "more" heavy-tailed than $Pareto_3$. We performed an analysis on a number of real Internet traces and we have not found a single trace that does not satisfy the Assumption 1. Assuming (6), for choosing c_0, we first estimate the number of flows of a single memory cell using bitmap technique[8](b is the size of the bitmap, we use $b = 200$; z is the number of zero bits):

$$\hat{n} = b \ln(\frac{b}{z}).$$

Then we use

$$c_0 = \bar{c}_0(Pareto_3, \hat{n}). \quad (7)$$

[12]Recall that choosing too large c_0 might cause the algorithm to stop searching for the heaviest flow by finding a flow (with rate $typRate$) that satisfies $\sqrt{\frac{eqPairs}{cRate}} \leq c_0 \frac{typRate}{cRate}$.

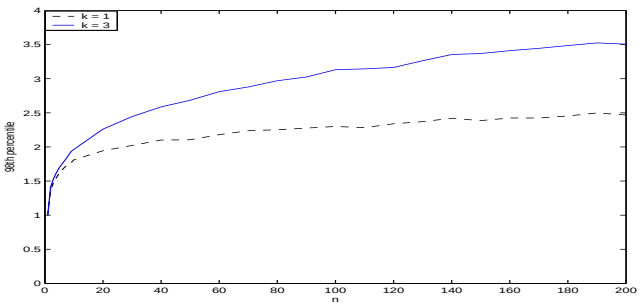

Figure 13: 98% percentile of the distribution of $\sqrt{\sum_i^n z_i^2}/\max(z_i)$, for $n = 1, ..., 200$. z_i are drawn with Pareto distribution with $k = 1$ (dashed line) and $k = 3$ (full line).

5. APPLICATION: HEAVY-HITTER HOLD - HH

In this section we present a queue management scheme called HH, that will be built on RHE. The main objective of HH is enforcing max-min fair bandwidth allocation. The reasons for employing "fair queueing" and its importance can be found in [33], [28] and [10]. Particulary, a queue scheme that enforce fair bandwidth allocation would protect responsive and not too aggressive flows from non-responsive and aggressive ones, and could be essential for providing QoS requirements once the number of users that use some of the high-speed [18, 12] or loss-insensitive transport protocols becomes large (compared to the total number of users).

The pseoudocode of HH is given in Figure 13. We use MIMD (Multiplicative Increase-Multiplicative Decrease) algorithm for controlling the variable $FAIR$ - that represents the maximal allowable rate: whenever the estimated rate, $estc$, of the flow the arriving packet belongs to, is greater than $FAIR$, the packet should be dropped with probability $(estc - FAIR)/estc$. We chose to update the variable $FAIR$ once in the interval of length δ, with same δ as in RHE, but of course the frequency of updating $FAIR$ and RHE parameters do not have to be same. Our performance goal is to keep utilization of the link at the prescribed value $util$. On each update of $FAIR$ we ask whether the utilization is less then $util$ or not and update $FAIR$ with the MIMD parameters $t_1 > 1$ and $t_2 > 1$.

Having information on $FAIR$, RHE will use threshold $T = FAIR/2$ rather than manually setting the threshold or self-tuning of the threshold. Setting the threshold on value $T = FAIR/2$ allows the monitoring of a long-lasting TCP flow after it responds to packet loss by halving its congestion window.

In the pseudocode given in Figure 13, RHEEstRate(pkt) and EstCELL(pkt) represent the estimated rate of the flow packet pkt belongs to and the memory cell that has given this estimate.

In order to prevent multiple losses for TCP (or more generally loss-responsive) flows, we introduced a 2bit variable $marked$ attached to each cell. For memory cell $CELL$, $marked(CELL)$ is 0 if no packet that corresponds to CELL has been dropped in both the current and previous update interval; $marked(CELL)$ is 1 if some packet that corresponds to CELL has been dropped in the current update interval but no packet that corresponds to CELL is dropped in the previous update interval; $marked(CELL)$ is 2 if some packet that corresponds to CELL has been dropped in the previous update interval but no packet that corresponds to CELL is dropped in current update interval and $marked(CELL)$ is 3 if some packets that correspond to CELL have been dropped in both the previous and the current update interval. If $marked(CELL)$ is equal to 1 no packets with EstCELL(pkt) equal to $CELL$ will be dropped until the next $FAIR$ update when $marked(CELL)$ is updated to reflect the definition we have already given.

```
On arrival of packet pkt
. [estc, CELL] = [RHEEstRate(pkt) EstCELL(pkt)]
.  if (estc < FAIR)
.     ShouldDrop ← 0;
.  else
.     rand = Random :: uniform(0, 1);
.     if (rand < (estc − FAIR)/estc)
.        ShouldDrop ← 1;
.     end if
.  end if
.  if marked(CELL) == 1
.     ShouldDrop ← 0;
.  end if
.  if ShouldDrop == 1
.     Drop(pkt);
.     if (marked(CELL) mod 2 == 0);
.        marked(CELL) ← marked(CELL) + 1;
.     end if
.  end if
.  if (now − LastUpdate > δ)
.     if (CurrentThroughput/Capacity < util)
.        FAIR = FAIR ∗ t1;
.     else
.        FAIR = FAIR/t2;
.     end if
.     LastUpdate = now;
.     for CELL = 1 : memorysize
.        if (marked(CELL) mod 2 == 1);
.           marked(CELL) ← 2
.        else if (marked(CELL) == 2)
.           marked(CELL) ← 0
.        end if
.     end for
.  end if
```

Figure 14: Pseudocode of HH

6. EVALUATION

Following the analysis from section 4.1 we set $\rho = 0.3$ in all our experiments[13]. Setting $\rho = 0.3$ implies $g(\rho) \approx 1$, meaning that number of cells that contain at least one of the largest $M/2$ flows is approximately equal to $M/2$, allowing high probability of identifying this top-$M/2$ flows. The parameter c_0 is tuned by rule (7).

6.1 Comparison between Multistage Filters, CATE and RHE

We will compare MSF[7], CATE[14], and RHE over 6 different synthetic streams where flow size distribution follows Pareto distribution with parameter $k = 0.5, 1, 1.5, 2, 2.5, 3$, and $m = 40$. Cumulative Distribution Functions (CDF) of this six distributions are shown in Figure 3.

For each k, length of stream is $L = 500000$, and the total number of flows is $NF = 10000$, each member of the stream has unit size and belongs to flow $i \in \{1, 2 \ldots, NF\}$ with probability w_i where w_i are chosen with Pareto distri-

[13]Code used in this section can be found at http://www.hamilton.ie/person/rade/RHE/.

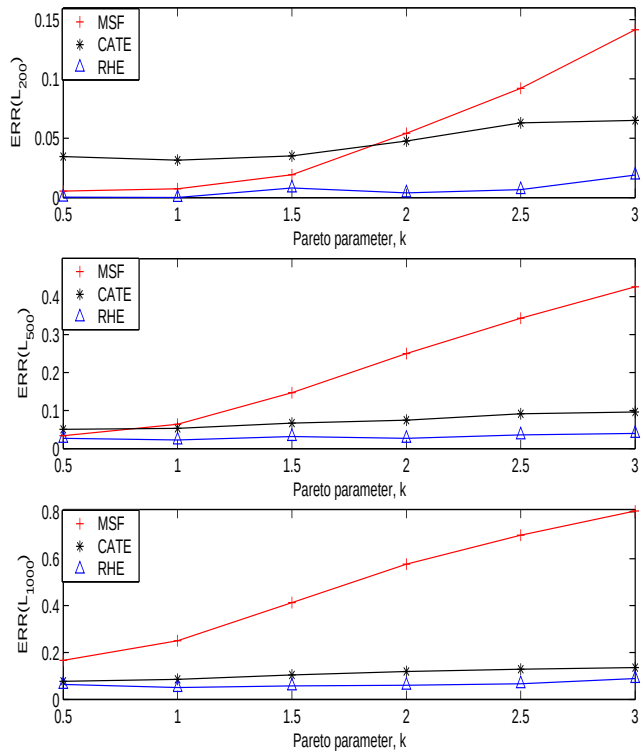

Figure 15: MSF vs CATE vs RHE. Average errors $ERR(L_{200})$, $ERR(L_{500})$ and $ERR(L_{1000})$ for different values of Pareto parameter.

bution with parameter k whose CDF is depicted in Figure 3. The parameters for weighted averaging of flow rates are $\delta = 0.5sec$, $q = 0.5$. Length of measurement interval for MSF is $\delta_1 = 2sec$. Number of arrivals per second is 50000, giving total length of single test 10 seconds. Available memory for RHE is $M = 1000$ memory cells. We assume that relative size of counters is 20% of a memory cell, and that relative size of flow entry (flow identifier + counter + pointer needed for hash table) is 40% of the size of a memory cell. For given memory of 1000 memory cells, values for the number of stages (d), the number of counters per stage (b) and the number of flow entries (FC) are derived from the recommendation given in Section 6.1 in [7]: $d = 4$, $b = 615$, $FC = 1270$. For CATE, we use $k = 50$ comparisons per packet[14], as recommended in [14].

The metric of interest is *average error between estimated traffic share and real traffic share* defined as in [7] by the ratio between the sum of the absolute values of the differences between the estimated and real traffic share divided by total sum of the real traffic share[15]. Thus, for a set of flows indexed by set I, the error is defined as:

$$ERR(I) = \frac{\sum_{i\in I} |est(f_i) - r(f_i)|}{\sum_{i\in I} r(f_i)} \qquad (8)$$

[14]Using more comparisons would be computationally expensive for high speed links($> 1Gbps$).

[15]RHE measures flow rates while CATE and MSF measure flow sizes so we use generic term "traffic share" to refer to flow rates in RHE case and flow sizes in MSF and CATE case.

We evaluated average errors for top 200, top 500 and top 1000 flows, indexed by sets L_{200}, L_{500} and L_{1000}. Our findings are presented in Figure 14. We observe that ratio between average errors of MSF and RHE grows as k grows, which shows much higher sensitivity of MSF on the "heavy-tail assumption". We also note low insensitivity of CATE on the "heavy-tail assumption": errors grow slowly with Pareto parameter k. However, high variance of CATE implies relatively large errors.

Comment 1. Note that the basic Multistage filtering algorithm (without shielding and conservative updates) identifies a flow as being large based on the information on $\min_i(AggrVol(C_i) \mid i = 1 : d)$. In the case of large number of flows per counter(say > 10) the quantity $AggrVol(C_i)$ will be (with high probability) within order of magnitude with volume of the heaviest flow that maps to that counter C_i only in the very heavy-tailed environments. Indeed, the ratio $AggrVol(C_i) \,/\, \max(Vol(f) \mid f \in C_i) = (\sum_{f\in C_i} z_f) / \max_{f\in C_i}(z_f)$ has a probability distribution that has a much larger mean than $\sqrt{(\sum_{f\in C_i}(z_f)^2)} / \max_{f\in C_i}(z_f)$ and therefore the quantity $\max_{f\in C_i}(Vol(f))$ is not dominant in $AggrVol(C_i)$ if either number of flows per counter is large or size distribution does not have *very* heavy tail. As we already noted in Section 2.1, the authors of MSF introduced multiple stages, shielding and conservative updates to improve confidence in finding heavy-hitters, while RHE uses additional information given in the memory cell to solve same problem.

Comment 2. Multiple stages in MSF are integral part of MSF algorithm, and a packet have to pass all stages twice (once for the counter increments and once for the conservative-update correction). However, RHE attempts to identify the heaviest flow at single stage and additional stages (with sizes that decrease geometrically) are introduced only because of possible collision when two heavy flows mapping to same cell. The following table shows average number of stages (ANS) per packet for RHE, for six different values of parameter k:

k	0.5	1	1.5	2	2.5	3
ANS	1.457	1.351	1.372	1.352	1.360	1.356

Comment 3. In [6] MSF is compared with a number of sketch based approaches (see [2, 21] and references therein) for identifying the heavy-hitters. It is claimed that MSF needs less memory to achieve given task. To quote [6]: *"...the conclusion is that multistage filters as I use them identify heavy hitters with less memory than sketches..."*.

6.2 Measurements on real packet traces.

In this section we present experimental results over 3 NLANR unidirectional traces [26] that we refer to as MRA, FRG and ALB. In all our experiments we use the definition of flow at the granularity of TCP connections, ie. a flow is defined by 5-tuple of source and destination IP address, and port and protocol numbers. We choose to compare measured and real rates at the moment $t_1 = 15sec$ from the start of measurement $t_0 = 0sec$. The number of flows, the number of packets and the average (total) throughput of these three traces in the interval $[t_0, t_1]$ are given in the following table:

Trace	# of flows	# of packets	Avg throughput
MRA	70826	905312	285.7 $Mbps$
FRG	27393	1030953	428.7 $Mbps$
ALB	181394	2501702	919.1 $Mbps$

Parameters used are: $\delta = 0.1sec$, $q = 0.9$, and the number of memory cells in MRA and FRG measurements is $M = 500$ and for larger ALB we set $M = 1000$.

In Table 1 we present our results. The metrics of interest are: proportion of identified $L_{M/5}$(top $M/5$ flows), $L_{M/2}$ and L_M flows(we denote proportion of identified L_k flows by $PropIden(k)$), the average error for $L_{M/5}$, $L_{M/2}$ and L_M flows and the average number of stages(ANS) per-packet. We also provide the average errors for L_M flows obtained by CATE (with $k = 50$ comparisons) and MSF with memory space equivalent to M memory cells(see previous subsection) on these three packet traces. Extremely large errors with CATE are due to relatively small sample (15 seconds of trace) and therefore high variance.

Metrics	MRA	FRG	ALB
$PropIden(M/5)$	0.99	1.00	1.00
$PropIden(M/2)$	0.904	0.896	0.960
$PropIden(M)$	0.648	0.694	0.730
$ERR(L_{M/5})$	0.0073	0.0122	0.0094
$ERR(L_{M/2})$	0.0277	0.0312	0.0241
$ERR(L_M)$	0.0409	0.0408	0.0412
ANS	1.5593	1.4057	1.5909
$ERR(L_M)$(MSF)	0.2837	0.1368	0.0625
$ERR(L_M)$(CATE)	0.3850	0.4780	0.4967

Table 1: Proportion of identified $L_{M/5}$, $L_{M/2}$ and L_M flows; average error for flows from $L_{M/5}$, $L_{M/2}$ and L_M; average number of stages(ANS) per-packet; MSF and CATE errors for L_M flows.

The average number of flows per memory cell on first stage is around 200 on MRA and ALB traces and around 80 on FRG trace. In spite of these large number of flows per memory cell, RHE is able to extract almost all the top-$M/5$ flows and about 90% of $M/2$ flows with highest rates. The difference between the number of identified $L_{M/2}$ and L_M flows and the upper bounds given in Section 4 is mainly caused by the fact that the threshold T often stabilizes around a value that does not allow for some non-identified flows from L_M and $L_{M/2}$ to pass from one stage to another. We should also observe results for the average number of stages per cell. On the most challenging trace ALB, with aggregate throughput of more than $900Mbps$, the average number of packets per second is around 166000. With 1.5909 stages per-packet, this gives more than $3\mu s$ per-packet update on one stage, defined on Figure 7. Assuming fast SRAM with access time of $4ns$[16], a packet update would take less than $100ns$, which is roughly 30 times less than that needed on trace ALB with aggregate throughput of about $919Mbps$.

For the trace A, let $k(A)$ be the largest integer such that all top-$k(A)$ flows are identified; we will denote by $TT(A)$ the proportion of traffic taken by top-$k(A)$ flows. On all three traces $TT(MRA), TT(FRG), TT(ALB)$ are greater than 60%. Thus RHE is able to *fully* identify top flows responsible for more than 60% of traffic on the gigabit links with only several kilobytes of SRAM memory. Figure 15 graphically illustrates the accuracy of RHE for all 3 traces.

6.3 Performance of HH.

In this subsection we will present ns2 packet level simulation results that will illustrate the behavior of HH. To do this we initiated 1000 TCP flows divided in three groups[17], whose TCP parameters are described in the Table 2 that share 40 Mbit/s link. We ran two experiments with these 1000 flows: one over the link with a Drop Tail queue and another over a HH queue. The queue size in both cases is 300Kbytes which is $60ms$ of buffering on $40Mbit/s$ link.

Flow ID (u)	$maxcwnd$	RTT (ms)	$packetSize$(B)
$u \in 1..20$	∞	$20 + 20u$	1500
$u \in 21..120$	30	$8u$	$650 - 5u$
$u \in 121..1000$	3	$8u + 40$	$60000/u$

Table 2: Characteristics of 1000 TCP connections.

HH parameters are $util = 0.98$, $t1 = 1.003$, $t2 = 1.003$. Weighting-average, parameters are the same as in the previous subsection $\delta = 0.1sec$, $q = 0.9$ and the number of memory cells is $M = 40$. Flows from the first group do not have maximal congestion window limitation and they account for 67.36% of bandwidth in the DropTail case. In the HH case, the total share of bandwidth for flows from the first group is reduced to 41.03%, while the total share of bandwidth for other two groups increased significantly as is sketched in Table 3 (here $TOT(a..b)$ denotes share of bandwidth taken by flows $a, a+1, \ldots, b$). The utilization in DropTail case is slightly higher than in the HH case which is almost equal to desired $util = 0.98$.

In the networking literature, the standard metrics used for evaluating the level of fairness for certain bandwidth allocations is Jain's fairness index(JFI)[16]. For a set of k numbers $a_1, \ldots, a_k$, the JFI is the square of the ratio between arithmetic and quadratic mean of these k numbers:

$$JFI(a_1, \ldots, a_k) = \frac{(a_1 + \cdots + a_k)^2}{k(a_1^2 + \cdots + a_k{}^2)}$$

The last row in Table 3 contains the JFI for share of bandwidth for flows from the first group in both the DropTail and HH case. Figure 16 graphically illustrates the share of bandwidth for flows from the first group in both scenarios.

Metrics	DropTail	HH
$TOT(1..20)$	0.6736	0.4103
$TOT(21..120)$	0.2933	0.5539
$TOT(121..1000)$	0.0330	0.0356
$Utilization$	0.9916	0.9799
Average queueing delay(ms)	43.7	10.9
$JFI(1..20)$	0.5706	0.9984

Table 3: Experimental results for DropTail and HH queue.

7. SUMMARY

In this paper we present a heuristic method named RHE for realtime identification of flows that account for most of the bandwidth. If positive numbers $z_1, \ldots, z_n$ are taken with some heavy tailed probability distribution then $\sum_{i=1}^n z_i^2$ is dominated by the maximum of z_i. Assuming that z_i are Internet flow rates, efficient estimation of $\sum_{i=1}^n z_i^2$ allows us to design an algorithm that with high confidence identifies the

[16]Current SRAM access time varies between $2ns$ and $5ns$[35].

[17]The first group mimics elephant flows, the second kangaroos and the third mice flows.

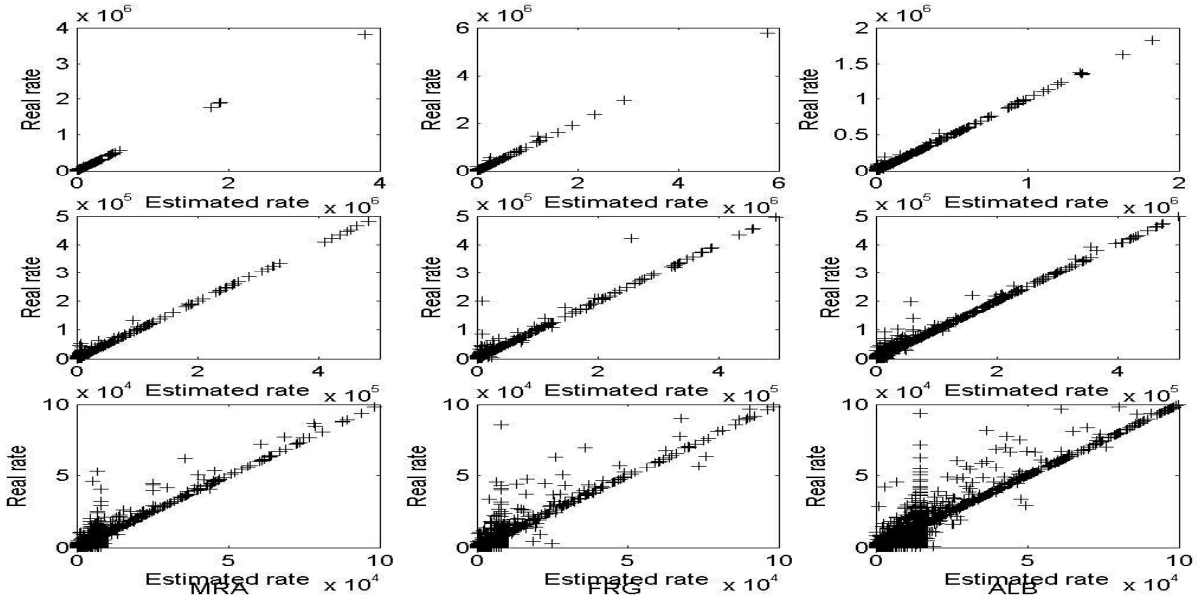

Figure 16: Estimated vs Real rates on MRA (left), FRG (middle) and ALB (right) trace. The top figures contains all flows, the middle figures zoom in on flows with rates $\leq 500KB/sec$ and the bottom zoom in on flows with rates $\leq 100KB/sec$

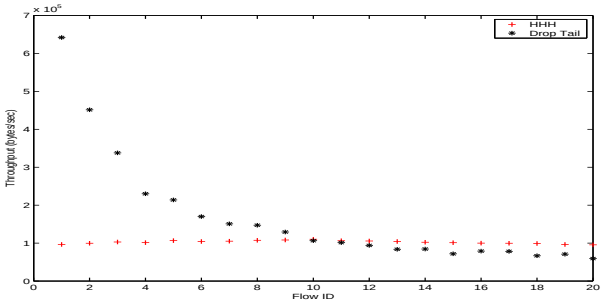

Figure 17: Throughput for flows from the first group under DropTail and HH.

highest rate flows. The presented method is highly scalable in the following sense(s)

- *Computationally.* Per packet processing requires only 3 comparisons, one hashing and up to 4 additions per stage. The design of RHE allows only a very few packets to higher levels and therefore the average number of stages per packet is just slightly above 1 (in all experiments presented here it is in range [1.3,1.6]). As a result of this, RHE can scale with line speeds of up to several tens of Gbps.

- *Memory space.* RHE needs a very small amount of memory. Experiments in Section 6.2 show high efficacy with just several kilobytes of SRAM memory on Gbps-traces. Low memory requirements of our scheme allow usage comparatively scarce on-chip SRAM which is highly desirable at highly loaded network processors.

- *Heavy-tail assumption.* RHE covers a wide range of flow rate distributions and is much more robust to heavy-tailed assumption, compared to MSF.

- *Number of active flows.*

Our main motivation for developing RHE was flow control. The proposed queue management scheme HH exploits the efficacy of RHE and controls fairness using a very small amount of memory. An important, highly nontrivial and open question that arises in the study of the "fair queueing" algorithms is:

(*) *What would the Internet flow rate distribution look like if routers employed fair queueing?*

In the context of a queueing scheme whose efficacy relies on the assumption of heavy-tailed traffic[18], the change in the flow rate distribution caused by fair queueing is an important metric that affects the performance of these algorithms. Our algorithms depend on the flow rate distribution only through the constant c_0 that defines the test given in Figure 5. Initial measurements indicate that Assumption 1 used for tuning c_0 is satisfied for all tested traces. However, analytical research is necessary to answer the question (*) en route to an efficient, scalable and robust "fair queueing" algorithm for the current and future Internet.

8. REFERENCES

[1] G. Appenzeller, I. Keslassy, N. McKeown. "Sizing router buffers". Proc. of the ACM SIGCOMM '04, Portland, Oregon, USA, 2004.

[2] L. Che, B. Qiu, "Landmark LRU: an efficient scheme for the detection of elephant flows at internet routers". IEEE Communićations Letters, vol. 10 (7), July, 2006.

[3] G. Cormode, S. Muthukrishnan. "What's hot and what's not: tracking most frequent items dynamically". ACM Transactions on Database Systems, Vol 30 , Issue 1, March 2005.

[4] A. Demers, S. Keshav, S. Shenker. "Analysis and simulation of a fair queueing algorithm". Proc. of the ACM SIGCOMM, Austin, TX, September 1989.

[18]The efficacy of most of Fair-queueing schemes depends on the flow rate distribution, see [20].

[5] D.L. Donoho. "De-Noising by Soft-Thresholding". IEEE Transactions on Information Theory, Vol. 41, No. 3, May 1995.

[6] N. Duffield. "Sampling for Passive Internet Measurement: A Review". Statistical Science, Volume 19, no. 3, 2004, Pages 472-498.

[7] C. Estan. "Comparison between multistage filters and sketches for finding heavy hitters". UCSD technical report CS2004-0784, April 2004

[8] C. Estan, G. Varghese. "New directions in traffic measurement and accounting". ACM Transactions on Computer Systems, Vol 21 , (3), 2003.

[9] C. Estan, G. Varghese, M. Fisk. "Bitmap algorithms for counting active flows on high speed links". Proc. of 3rd ACM SIGCOMM conference on Internet measurement. Miami Beach, Florida, USA, 2003.

[10] C. Estan, G. Varghese. "Building a better NetFlow". Proc. of ACM SIGCOMM '04, Portland, Oregon, USA, 2004.

[11] K. Fall, S. Floyd. "Router mechanisms to support end-to-end congestion control". [online] ftp://ftp.ee.lbl.gov/papers/collapse.ps.

[12] S. Floyd. "HighSpeed TCP for Large Congestion Windows". RFC 3649, Experimental, December 2003.

[13] F. Hao, M. S. Kodialam, T. V. Lakshman. "ACCEL-RATE: a faster mechanism for memory efficient per-flow traffic estimation". ACM SIGMETRICS '04, New York, NY, USA, 2004.

[14] F. Hao, M. Kodialam, T. V. Lakshman, H. Zhang. "Fast, Memory-Efficient Traffic Estimation by Coincidence Counting". Proc. of IEEE INFOCOM 2005, Miami, Florida, USA, 2005.

[15] N. Hohn, D, Veitch. "Inverting sampled traffic". Proc. of 3rd ACM SIGCOMM conference on Internet measurement. Miami Beach, Florida, USA, 2003.

[16] R. Jain. "The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling". John Wiley and Sons, INC., 1991.

[17] D. Katabi, M. Handley, C. Rohr. "Internet Congestion Control for Future High Bandwidth-Delay Product Environments". Proc. of ACM SIGCOMM, Aug. 2002, Pittsburgh, PA.

[18] T. Kelly. "Scalable TCP: Improving Performance in Highspeed Wide Area Networks". ACM SIGCOMM Computer Communication Review Volume 33 , Issue 2, April 2003.

[19] M. Kodialam, T. V. Lakshman, S. Mohanty. "Runs bAsed Traffic Estimator (RATE): A Simple, Memory Efficient Scheme for Per-Flow Rate Estimation". Proc. of IEEE INFOCOM 2004, Hong Kong, China, 2004.

[20] A. Kortebi, L. Muscariello, S. Oueslati, J. W. Roberts. "Evaluating the number of active flows in a scheduler realizing fair statistical bandwidth sharing". ACM SIGMETRICS '05, Banff, Canada, 2005.

[21] B. Krishnamurthy, S. Sen, Y. Zhang, Y. Chen. "Sketch-based change detection: methods, evaluation, and applications". Proc. of the ACM SIGCOMM conference on Internet measurement. Miami Beach, Florida, USA, 2003.

[22] A. Kumar, J. Xu, L. Li, J. Wang. "Space-code bloom filter for efficient traffic flow measurement". Proc. of IEEE INFOCOM 2004, Hong Kong, China, 2004.

[23] A. Kumar, J. Xu, L. Li, J. Wang. "Data streaming algorithms for efficient and accurate estimation of flow size distribution". ACM SIGMETRICS '04, New York, NY, USA, 2004.

[24] D. Lin, R. Morris. "Dynamics of random early detection". Proc. of the ACM SIGCOMM '97, Cannes, France, 1997.

[25] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, S. Shenker. "Controlling high bandwidth aggregates in the network". ACM SIGCOMM Computer Communication Review Volume 32, Issue 3, Pages: 62 - 73, July 2002.

[26] NetFlow documentation, available online: http://www.cisco.com/warp/public/732/Tech/nmp/netflow/

[27] NLANR IP traces, online, MRA (ftp://pma.nlanr.net/traces/daily/20050416/MRA-1113621669-1.tsh.gz), FRG (http://pma.nlanr.net/Traces/Traces/old/20011015/FRG-1114614867-1.tsh.gz), ALB(ftp://pma.nlanr.net/traces/long/ipls/1/IPLS-CLEV-20020814-103000-0.gz).

[28] T. J. Ott, T. V. Lakshman, L. H. Wong. "SRED: Stabilized RED". Proc. of IEEE INFOCOM, New York, 1999.

[29] R. Pan, L. Breslau, B. Prabhakar, S. Shenker. "Approximate fairness through differential dropping". ACM SIGCOMM Computer Communication Review Volume 33, Issue 2, Pages: 23 - 39, April 2003.

[30] R. Pan, B. Prabhakar, K. Psounis. "CHOKe a stateless active queue management scheme for approximating fair bandwidth allocation". Proc. of IEEE INFOCOM 2000, Tel Aviv, Israel, 2000.

[31] S. Ramabhadran, G. Varghese. "Efficient implementation of a statistics counter architecture". ACM SIGMETRICS '03, San Diego, CA, USA, 2003.

[32] M. Shreedhar, G. Varghese. "Efficient fair queueing using deficit round-robin". *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, June 1996.

[33] R. Srikant, *The Mathematics of Internet Congestion Control*. Birkh̀auser, 2004.

[34] R. Stanojevic. "Small active counters". Technical report, available online: http://www.hamilton.ie/person/rade/counters.pdf.

[35] I. Stoica, S. Shenker, H. Zhang. "Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks". *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, 33-46, February 2003.

[36] Q. Zhao, J. Xu, Z. Liu. "Design of a Novel Statistics Counter Architecture with Optimal Space and Time Efficiency". ACM SIGMETRICS '06, France, 2006.