Identifying High Cardinality Internet Hosts

Jin Cao*, Yu Jin†, Aiyou Chen*, Tian Bu* and Zhi-Li Zhang†
* Bell Laboratories, Alcatel-Lucent † Computer Science Dept., University of Minnesota

***Abstract*—The Internet host cardinality, defined as the number of distinct peers that an Internet host communicates with, is an important metric for profiling Internet hosts. Some example applications include behavior based network intrusion detection, p2p hosts identification, and server identification. However, due to the tremendous number of hosts in the Internet and high speed links, tracking the exact cardinality of each host is not feasible due to the limited memory and computation resource.**

Existing approaches on host cardinality counting have primarily focused on hosts of extremely high cardinalities. These methods do not work well with hosts of moderately large cardinalities that are needed for certain host behavior profiling such as detection of p2p hosts or port scanners. In this paper, we propose an online sampling approach for identifying hosts whose cardinality exceeds some moderate prescribed threshold, e.g. 50, or within specific ranges. The main advantage of our approach is that it can filter out the majority of low cardinality hosts while preserving the hosts of interest, and hence minimize the memory resources wasted by tracking irrelevant hosts. Our approach consists of three components: 1) two-phase filtering for eliminating low cardinality hosts, 2) thresholded bitmap for counting cardinalities, and 3) bias correction. Through both theoretical analysis and experiments using real Internet traces, we demonstrate that our approach requires much less memory than existing approaches do whereas yields more accurate estimates.

I. Introduction

The Internet host *cardinality*, defined as the number of peers that a host communicates with, is an important metric for host profiling. Identifying hosts with high cardinalities in specific ranges is useful for many network operations such as traffic classification and intrusion detection, as they are often associated with various network events of interest. For instance, when a worm scans the Internet to propagate itself, we can observe traffic originated from the IP address hosting the worm towards a large number of peers. As another example, hosts running p2p applications usually originate and receive traffic from a *moderately* large number of peers.

One major challenge for identifying hosts with high cardinalities in specific ranges is the scalability problem due to the extraordinary number of hosts in the high speed Internet, which makes it either too slow or too expensive to track the statistics of individual host exactly. This is especially true when the algorithm needs to be implemented inside the router or firewall with tight memory constraints. In addition, the identification algorithm is often required to work in the online setting due to the huge traffic volume on high speed Internet links, i.e. on a sequence of source-destination pairs extracted from the traffic stream[1], where each record can only be observed once.

Being aware of the fact that most of the host profiling applications can work just fine with approximated statistics, existing work in literature have been proposed on using sampling based method to identify *superspreaders*, i.e., hosts with very large cardinalities defined by a threshold t. By using a simple pair-based sampling[2], *only* source hosts of the sampled pairs will be tracked for their cardinalities [15], [18]. It is easy to show that, with a pair sampling rate of r, the chance that a host of cardinality N whose corresponding pairs will be sampled at least once is $p = 1-(1-r)^N$. For example when the cardinality is 1, the probability is only r.

Although a simple pair-based sampling maybe sufficient for eliminating hosts with low cardinalities in the identification of super-spreaders, it is not sufficient for applications where the thresholds are much smaller such as detecting p2p hosts (e.g., cardinality within [50, 500]), as too much memory are wasted to track hosts with low cardinalities. This is a result of two factors: a larger pair sampling rate r (so that hosts with cardinalities above t would not be erroneously eliminated), and the Zipf or power law distribution of the number of hosts with small cardinalities as evident in real Internet traces (see Fig. 1(a)). Fig. 1(b) demonstrate this phenomenon by showing the cumulative probabilities of the host cardinalities from sampled pairs, for the sampling rates $r = 0.1, 0.01, 0.001$. For instance, when $r = 0.01$, more than 75% hosts that are being tracked has a cardinality less than 20.

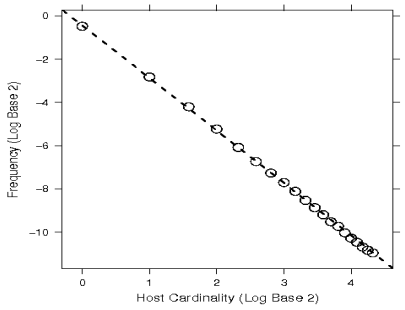
A. Overview of Our Approach

In this paper, we present a new algorithm which is capable of identifying high cardinality hosts defined by a *moderately large* threshold t or within predefined threshold ranges. Our main idea is to use a *two-phase filtering* method for eliminating the majority of low cardinality hosts, so that we can reserve more resource for each remaining candidate to obtain a more accurate estimate. Hence our method only needs a fraction of memory as compared to existing approaches, and is not limited to very high cardinality thresholds as previous approaches do.

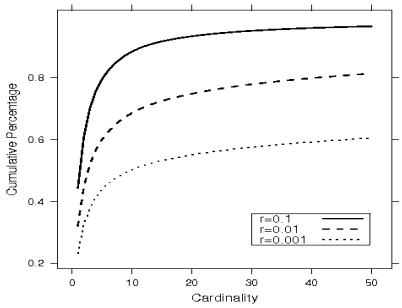
A schematic representation of our overall approach is shown in Fig. 2. Given a network stream, we first apply a two-phase filtering (pre-filtering and parallel filtering with K dependent pair samplers) method to eliminate the majority of low cardinality hosts. Only those hosts that have passed the two-phase filtering will become candidates and are subject to

[1]A traffic stream can be a packet trace or a flow (aggregated packets, e.g. NetFlow) trace. Each pair consists of the source and destination addresses of the corresponding packet/flow. Notice that pairs may duplicate in the trace.

[2]Pair-based sampling is referred to as distinct sampling on pairs, i.e. we sample a pair f if $(hash(f) \mod C)/C < r$, where C is a large constant and r is the sampling rate. Using pair-based sampling, only unique pairs are considered during cardinality counting.



(a) Frequency distribution of the host cardinalities. Notice both axes are in log scale, and hence that the probability distribution is Zipf-like. The dotted line is a linear regression curve for the frequencies.



(b) Cumulative probabilities of the host cardinalities from sampled source-destination pairs, with the sampling rate $r = 0.1, 0.01, 0.001$, respectively.

Fig. 1. Characteristics of a university trace TR_1 with 6.14 million hosts, which is studied in detail in Section IV. See Table I for a detailed description.

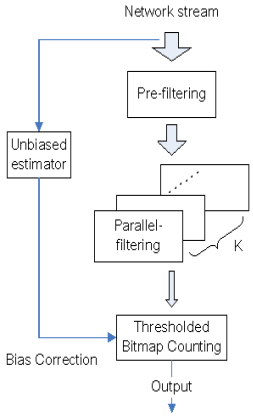


Fig. 2. A schematic representation of our approach.

further cardinality counting. We then apply a memory efficient thresholded bitmap algorithm for estimating the cardinalities of candidate hosts and thereafter identify high cardinality ones above threshold t or within specific ranges. In addition, we notice that, in the online setting, our cardinality estimates could be biased as there might be lost pairs for each candidate host due to the filtering and counting process. To account for the lost pairs, we obtain unbiased cardinality estimates simultaneously for a small random sample of hosts, and use the average number of missed pairs of those hosts which have both kinds of cardinality estimates for bias correction.

Our main contribution can be summarized as follows:

1) We develop a memory efficient, sampling based, two-phase filtering approach to eliminate the majority of low cardinality hosts.

2) We propose a novel *dependent pair sampling* scheme that we use in the two-phase filtering to significantly reduce low cardinality hosts. We analyze the sampling performance theoretically, and demonstrate the effectiveness of dependent pair sampling over independent pair sampling.

3) We develop a thresholded bitmap algorithm to estimate the cardinalities of candidate hosts after they passed the two-phase filtering. Our method extends the work in [17], and works especially well at identifying hosts with cardinalities above a moderately large threshold or within specific ranges.

4) We derive an estimate of the average number of lost pairs for high cardinality hosts, due to two-phase filtering and thresholded bitmap. Such estimate requires very little extra memory, and can correct the bias in the original estimates. To our knowledge, we are the first to address this bias issue.

5) We demonstrate through experiments using real Internet traces that our method requires much less memory than previous approaches do whereas yields more accurate estimates.

The rest of the paper is organized as follows. In Section II, we introduce a memory-efficient two-phase filtering scheme to eliminate the majority of low cardinality hosts. In Section III we discuss the thresholded bitmap algorithm for cardinality estimation and present our bias correction method. We show the effectiveness of our approach with experiments on real Internet traces in Section IV. Finally, we discuss other related work in Section V and conclude the paper in Section VI.

II. Two-Phase Filtering for Eliminating Low Cardinality Hosts

In this section, we describe a two-phase filtering approach for eliminating the majority of low cardinality hosts: only hosts that have passed this stage are selected for further counting.

A. Two-Phase Filtering

The essence of our two-phase filtering process is to *select hosts by sampling pairs* and to increase the filtering power toward low cardinality hosts by combining multiple pair samplers. To select hosts by sampling pairs, for each pair sampler, we record a host at the first time that one of its pairs is sampled, and let the remaining pairs from that host (if any) pass the sampler directly. Only hosts that are selected by all pair samplers will become candidates for further counting.

Our two-phase filtering approach includes a *pre-filtering* step and a *parallel filtering* step. The pre-filtering step consists of a single pair sampler with rate r_0, which is designed to effectively shrink the size of candidate hosts before the parallel filtering step (to remove hosts with very low cardinalities, e.g. $\leq 1/r_0$). As a result, the total required memory is dramatically reduced[3]. Pairs passing through the pre-filtering step are then sent to the parallel filtering step, where the candidate hosts are further reduced by using K parallel pair samplers, each with a pair sampling rate r. We assign a Bloom Filter [3] to each of the $K+1$ pair sampler to record the sources hosts of those sampled pairs. Pairs which have survived both filtering steps (i.e. their source addresses present in all $K+1$ Bloom Filters) are subject to further counting.

The idea of using $K, K \geq 2$, parallel pair samplers for eliminating low cardinality hosts comes from the following observations[4]. Given a host with cardinality N, the probability that at least one of its pairs will be sampled by one pair sampler with rate r is $p_1(N) = 1 - (1-r)^N$. With K

[3]Due to space limitation, we do not demonstrate it here, but refer the readers to the extended version of the paper[1]

[4]To analyze the host selection probability for parallel sampling, we temporarily ignore the effect of false positives caused by the Bloom Filter.

independent pair samplers, the probability that the host is selected by all K samplers is now reduced to

$$p_K(N) = p_1(N)^K = (1-(1-r)^N)^K \quad (1)$$

which is significantly less for a low cardinality host with a small $p_1(N)$. On the contrary, for a high cardinality host whose $p_1(N)$ is close to 1, $p_K(N)$ will remain close to 1. As an example, assume $K = 2$ and $r = 0.1$. Now for a host with cardinality 1, the probability that it will be selected is reduced to $p_2(1) = r^2 = 0.01$ from 0.1 of using one pair sampler. For a host with cardinality 2, the probability is reduced to $p_2(2) = (2r - r^2)^2 = 0.036$ from 0.19. But when $N = 50$ such that $p_1(N) = 0.995$, $p_2(N) = 0.99$. Realizing that in typical Internet traffic traces, the number of low cardinality hosts dominates, thus, using K parallel pair samplers will significantly reduce the number of low cardinality hosts.

Though independent sampling method is capable of reducing low cardinality host, in this paper, we propose an alternative dependent sampling method which is computationally cheaper and better at eliminating low cardinality hosts.

The new dependent sampling approach is presented as follows. Assume $Kr \leq 1$[5]. For each pair, first we sample it with a rate Kr. For each sampled pair, we then hash it uniformly to an integer i between 1 and K and record the source host of that sampled pair in the ith Bloom Filter, $1 \leq i \leq K$. Now it is easy to see that each pair sampler i still has a sampling rate of $Kr \times 1/K = r$. The difference of this from using K independent pair samplers is that now sampled pairs among the K samplers are negatively correlated, as each pair can only be sampled once by the K parallel samplers. Therefore, a host that appears in all K Bloom Filters will have a cardinality of at least K, which implies that we will eliminate all hosts that have cardinalities smaller than K. Compared to Eq. 1 for K independent sampling, we have the following results for K dependent sampling:

Theorem 1: For a host with cardinality N, the host sampling probability for K dependent pair samplers is:

$$p_K(N) = \sum_{j=0}^{K} \binom{K}{j} (-1)^j (1-jr)^N \quad (2)$$

Furthermore, for the same values of K, N such that $K, N > 1$, it can be shown that $p_K(N)$ is always less for dependent sampling than independent sampling.

The proof is provided in the appendix. For dependent sampling, when $K \leq 4$, plugging into Eq. (2), we have

$$\begin{aligned} p_1(N) &= 1-(1-r)^N, \\ p_2(N) &= 1-2(1-r)^N + (1-2r)^N, \\ p_3(N) &= 1-3(1-r)^N + 3(1-2r)^N - (1-3r)^N \\ p_4(N) &= 1-4(1-r)^N + 6(1-2r)^N - 4(1-3r)^N \\ &\quad + (1-4r)^N \end{aligned} \quad (3)$$

Fig. 3 plots the host selection probability against its cardinality, for both kinds of parallel sampling with $r = 0.1$, and $K \leq 4$. It is easy to see that the rise of host selection probabilities is fairly fast when the cardinality increases. For large cardinalities (say beyond 40), all selection probabilities are close to 1 (It is easy to show that when the cardinality N approaches infinity, the difference between independent and dependent sampling converges to 0). However, for small cardinalities, the selection probabilities are much less using dependent sampling method than those using independent method. In other words, the dependent sampling approach is capable of eliminating more low cardinality hosts.

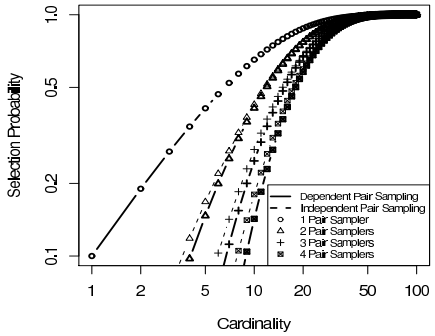


Fig. 3. Host selection probability vs. cardinality.

Another benefit from the dependent sampling scheme is the increase of per-pair processing speed. For each pair, the K independent sampling requires K hash operations. Instead, only one hash computation is needed using K dependent samplers. Hence, in our subsequent experiments, we focus on dependent sampling approach due to its superior performance.

Algorithm 1 Two-phase Filtering

```
Parameters: Pair sequence F_0, r_0, r, K, C;
Initialization: K+1 Bloom Filters BF_0, ..., BF_K, hash table HF.
for each f ∈ F_0 do
    Extract srcAddr, dstAddr from f;
    if srcAddr ∈ HF then
        Let f pass the two-phase filtering for cardinality estimation;
    else
        Generate a random number p = (hash(f) mod C)/C;
        if srcAddr ∉ BF_0 and p ∈ [0, r_0) then
            Insert srcAddr into BF_0;
        else if srcAddr ∈ BF_0 and p ∈ [r_0, r_0 + K × r) then
            Insert srcAddr into BF_i, i = ⌈r^{-1}(p − r_0)⌉;
            if srcAddr appears in all BF_j, j = 1, ..., K then
                Add srcAddr to hash table HF;
```

A detailed description of our two-phase filtering is presented in Algorithm 1. We use a hash table HT to store all candidate hosts that have passed the two-phase filtering and are subject to counting[6]. Assuming that $r_0 + Kr < 1$[7]. For each incoming pair f with source host address $srcAddr$, first we check if $srcAddr$ is in table HT. If yes, we let it directly pass the two-phase filtering and start counting its cardinality. Otherwise, we check if $srcAddr$ is present in the Bloom Filter BF_0 to see whether that host has already passed the pre-filering step. Meanwhile, we generate a uniform random number $p \in [0, 1]$ based on f. If $srcAddr$ is not in BF_0 and $p \leq r_0$, this implies

[5]The scheme can be adapted easily when $Kr > 1$ by splitting Kr into multiples of r that are smaller than 1.

[6]The hash table HT will be further used during counting step and it facilitates the process for checking sampled hosts.

[7]The scheme can be easily adapted otherwise by splitting $r_0 + Kr$ into unit of 1s.

that f is sampled by the pre-filtering, so we record $srcAddr$ in BF_0. If $srcAddr$ is in BF_0 and $r_0 \leq p \leq r_0 + Kr$, this implies that f is sampled by the ith of the K dependent pair samplers, where $i = \lceil (p - r_0)/r \rceil$. We store $srcAddr$ in BF_i and check whether or not $srcAddr$ is present is all K Bloom Filters $BF_j, 1 \leq j \leq K$. If so, we add it to HT and start counting its cardinality in the remaining trace. Notice that since the sampling rate for all the K parallel pair samplers are the same, we can allocate the same amount of memory for all the Bloom Filters. In addition, we only need two sets of hash functions for Bloom Filter implementation, one for BF_0, the other can be shared by $BF_j, 1 \leq j \leq K$.

B. Number of Unique Pairs in Filtering

In the online setting, we only see each record once. Therefore, any pair entering the two-phase filtering may never appear later for cardinality counting, which will result in estimation bias. For a host with cardinality N, let $L_K(N)$ be the number of distinct pairs observed in the trace until the host successfully passes the two-phase filtering. Obviously, we do not like a large value of $L_K(N)$ with respective to N, since a large $L_K(N)$ is likely to correspond to a high bias (we call it filtering bias). Because it is very difficult to track the exact value of $L_K(N)$, in this section, we derive bounds for $L_K(N)$.

It is obvious that $L_K(N) \leq \min(N, L_K(\infty))$, where $L_K(\infty)$ is the number of distinct pairs seen in the two-phase filtering before a host with infinite number of pairs passes the filtering process. Notice when N is large enough such that the host selection probability is close to 1, $L_K(N) \approx L_K(\infty)$. To simplify the notation, let us redefine $L_K \equiv L_K(\infty)$. The following discussions now focus on L_K.

For a host with infinite pairs, let $L_K^{(0)}$ and $L_K^{(1)}$ be the number of distinct pairs seen during pre-filtering and parallel filtering, respectively. We have the following two results.

Corollary 1: For $n \geq 1$,

$$\begin{aligned} P(L_K^{(0)} = n) &= (1 - r_0)^{n-1} - (1 - r_0)^n, \qquad (4) \\ P(L_K^{(1)} = n) &= \sum_{j=0}^{K} \binom{K}{j} (-1)^{j+1} jr(1 - jr)^{n-1}, \end{aligned}$$

and

$$\begin{aligned} E(L_K^{(0)}) &= r_0^{-1}, \qquad (5) \\ E(L_K^{(1)}) &= \sum_{j=1}^{K} (-1)^{j+1} \binom{K}{j} (j^{-1} r^{-1} - 1). \end{aligned}$$

The proof utilizes the following two facts. First notice that $L_K^{(1)} = n$ implies that the host finally appears in all K Bloom Filters after n but not $n - 1$ pairs, this implies that

$$P(L_K^{(1)} = n) = p_K(n) - p_K(n - 1). \qquad (6)$$

Second, notice from Eq. (6), the selection probability $p_K(n)$ of a host with cardinality n is 1 minus the cumulative distribution of $L_K^{(1)}$, and therefore, $E(L_K^{(1)}) = \sum_{n=1}^{\infty}(1 - p_K(n))$. Now the corollary can be easily shown applying Eq. (2).

Lemma 1: $\max(L_K^{(0)}, L_K^{(1)}) \leq L_K \leq L_K^{(0)} + L_K^{(1)}$, so

$$E(\max(L_K^{(0)}, L_K^{(1)})) \leq E(L_K) \leq E(L_K^{(0)}) + E(L_K^{(1)}). \qquad (7)$$

Furthermore, the cumulative distribution of L_K can be bounded by that of $\max(L_K^{(0)}, L_K^{(1)})$ and $L_K^{(0)} + L_K^{(1)}$ for independent $L_K^{(0)}$ and $L_K^{(1)}$, which can be computed using (5).

For the host with infinite pairs, let $\mathcal{L}_K, \mathcal{L}_K^{(0)}$ and $\mathcal{L}_K^{(1)}$ be the set its pairs observed in the two-phase filtering, pre-filtering and parallel filtering, respectively. The lemma can be easily derived using the simple set relation: $\mathcal{L}_K = \mathcal{L}_K^{(0)} \cup \mathcal{L}_K^{(1)}$. The reason that we can only derive bounds regarding to L_K is because we do not know how the distinct pair sets $\mathcal{L}_K^{(0)}$ and $\mathcal{L}_K^{(1)}$ interact with each other. Think of two scenarios. In the first scenario, pairs of a source host appear in order, i.e., duplicates from 1st unique pair, duplicates from 2nd unique pair, and so on. In this case, $\mathcal{L}_K^{(0)}$ and $\mathcal{L}_K^{(1)}$ are exclusive. In the second scenario, each distinct pair duplicates infinite number of times. In this case $\mathcal{L}_K^{(0)} \subset \mathcal{L}_K^{(1)}$.

III. Counting Cardinalities of Candidates

Once a host has passed the two-phase filtering and becomes a candidate, we start counting its cardinality. In this section, we firstly present the thresholded bitmap algorithm for cardinality estimation. In the later part of this section, we derive an estimate of the average missed pairs which we will use for correcting bias in cardinality estimates.

A. Range Estimation Using Thresholded Bitmap

To identify high cardinality hosts defined by threshold t, we use the virtual bitmap algorithm proposed in [9]. It is established in [9] that for a target cardinality value t, the virtual bitmap is the least memory consuming scheme for counting cardinalities among the competing algorithms, with an optimal bitmap sampling rate $f_1 = 1.594/n$. Analysis shows that the relative accuracy of the virtual bitmap estimate changes less on the right-side of the optimal sampling rate compared to the left-side. To account for the missed pairs of candidate hosts due to the two-phase filtering in the online setting, we use a somewhat smaller virtual bitmap sampling rate

$$f_1 = 1.594/(t - E(L_K^{(1)}) - E(L_K^{(0)})), \qquad (8)$$

where $E(L^{(0)}), E(L^{(1)})$ can be computed using Eq. (5), and from Lemma 1, $L_K^{(0)} + L_K^{(1)}$ is the maximum number of missed pairs due to the two-phase filtering.

Although a single virtual bitmap is sufficient to identify high cardinality hosts, the cardinality estimates tend to be inaccurate if they fall outside of a narrow range around t [9], [17]. In the following, we present an extension of the scaled bitmap algorithm proposed in [17], which we refer to as the *thresholded bitmap* algorithm, and is designed to give reasonably good estimates of the *cardinality ranges*. It has been established in [17] that it uses very little extra memory compared with a single virtual bitmap, and is much more memory efficient than the competing algorithms (e.g. multi-resolution bitmap[9] and the Super-Loglog algorithm [8]).

Our thresholded bitmap algorithm is explained as follows (Algorithm 2). Suppose $N_{\max}$ is the largest possible cardinality value. Split the interval $[t, N_{\max}]$ into logarithmically

equal size intervals using a multiplying factor c: $[t, ct), [ct, c^2t)$ and so on. Suppose there are I such intervals, and denote the boundary of the intervals be $t_i, 1 \leq i \leq I$ so that $t_1 = t, t_2 = ct$ and so on. Define $f_i, i = 2, \ldots, I$ be

$$f_i = 1.594/(t_i - t_{i-1}). \tag{9}$$

Notice that f_i is the optimal virtual bitmap sampling rate for the target cardinality t_i. Now we run the virtual bitmaps corresponding to rates $f_i, 1 \leq i \leq I$ (note that f_1 is defined in Eq. (8)), in sequence as follows. For each candidate host, we start with the virtual bitmap that corresponds to the first sampling rate f_1. At any point if the cardinality estimate exceeds t_1, we set all bitmap bits to 0 and start with the next virtual bitmap in sequence, which in this case is the one that corresponds to sampling rate f_2. And then start the virtual bitmap with sampling rate f_3 if the estimate exceeds t_2, and so on. For a host, if the last bitmap corresponds to the sampling rate f_i, and let $\hat{N}^{last}$ be the cardinality estimate from the last virtual bitmap, then the cardinality estimate of the host is

$$\widehat{N} = \max(t_{i-1}, \hat{N}^{last}) \tag{10}$$

It is clear that to implement this, we do not consume additional space for the newer bitmaps, and all we need is to add some extra bits ($\log(I)$) to indicate the sequence numbers of the bitmaps. In practice, we have found that our method works well for any multiplying factor c that is larger than 5. However, since we reset the bitmap bits to zero if the cardinality estimate crosses thresholds t_i, pairs seen before the reset may never appear afterward. Those missed pairs lead to the second bias of estimation, and we call counting bias. Fortunately, due to the logarithmic spacing of the thresholds, the lost pairs in the online setting is only a small proportion when compared to the thresholds. In addition, in the next section, we will develop a bias correction method to address both the filtering bias and the counting bias.

Algorithm 2 Thresholded Bitmap

```
Parameters: Pair sequence F after filtering, t_i, 0 ≤ i ≤ I, M;
Initialization: Sampling rates f_i by Eq. (8) and (9).
//Cardinality counting
for each f ∈ F do
    Extract srcAddr, dstAddr from f;
    if srcAddr has not been recorded then
        Assign srcAddr with level 1;
        Allocate a bitmap with size M for srcAddr;
    Identify the level i of srcAddr;
    Update the corresponding bitmap with sampling rate f_i;
    if bitmap estimate exceeds t_i then
        Upgrade srcAddr to level i + 1, reset its bitmap;
//Estimation
for each srcAddr recorded do
    Identify the level L of srcAddr;
    Count nonempty entries in the bitmap: Y;
    N̂^last = log(1 − Y/M)/ log(1 − f_L)); (from [9])
    N̂_srcAddr = max(t_{L−1}, N̂^last);
```

B. Estimating the Number of Missed Pairs

Let b_h be the number of missed pairs for host h as a result of filtering bias and counting bias. As we know it is difficult to obtain b_h exactly for each host. Now consider a group of hosts $\mathcal{H}$. For each host $h \in \mathcal{H}$, denote its cardinality by N_h, and its biased estimate from the proposed procedure by $\widehat{N}_h^{biased}$. Let

$$\bar{b} = \sum_{h \in \mathcal{H}} (\widehat{N}_h^{unbiased} - N_h)/|\mathcal{H}|$$

be the average bias of b_h for $h \in \mathcal{H}$. In the following, we shall show that it is possible to obtain an accurate estimate of $\bar{b}$ under reasonable senarios. If so, we will be using $\bar{b}$ to correct the bias for hosts in $\mathcal{H}$ so that even though for each host it may be biased, as a group the average bias will be 0.

Our approach works as follows. For each host, in addition to the biased estimate $\widehat{N}_h^{biased}$, we obtain an unbiased estimate $\widehat{N}_h^{unbiased}$ using the pair sampling method in [15][8]. Denote the average difference between two cardinality estimates by

$$\widehat{b} = \sum_{h \in \mathcal{H}} (\widehat{N}_h^{biased} - \widehat{N}_h^{unbiased})/|\mathcal{H}| \tag{11}$$

Let $N_{\mathcal{H}} = \sum_{h \in \mathcal{H}} N_h$, and let q be the pair sampling rate that is used to derive $\widehat{N}_h^{unbiased}$ using the method in [15]. The following lemma states that $\hat{b}$ is an accurate estimate of $\bar{b}$ when $qN_{\mathcal{H}} \to \infty$ is large enough.

Lemma 2: If $qN_{\mathcal{H}} \to \infty$, then $\widehat{b} - \bar{b} \to 0$ almost surely.

Lemma 2 can be easily shown due to the Poisson approximation of $\sum_{h \in \mathcal{H}} \widehat{N}_h^{unbiased}$ (with mean $qN_{\mathcal{H}}$) and the law of large numbers. A consequence of Lemma 2 is that we can use a cheap memory to derive the unbiased estimates $\widehat{N}_h^{unbiased}$ using the pair sampling technique in [15], so that even though each estimate of $\widehat{N}_h^{unbiased}$ maybe poor, as long as $qN_{\mathcal{H}}$ is large enough, $\widehat{b}$ is still an accurate estimate of $\bar{b}$. For instance, we only need $qN_{\mathcal{H}} = 1000$ to have the bias accurately estimated within 3%.

We implement the bias correction using the average bias estimates as follows. First, to further reduce memory, we select a representative sample of hosts in $\mathcal{H}$ using host sampling with probability s, and obtain unbiased cardinality estimates only for those selected hosts using method proposed in [15]. Second, to improve the effectiveness of the bias correction using the average bias in (11), we compute the average bias for all hosts in the estimated cardinality range $[t_i, t_{i+1})$, $i = 1, \cdots, I$, and use it to correct the bias of all hosts in the range.

The host sampling rate s and the pair sampling rate q are selected based on the following considerations. First, s cannot be too small in order to get hosts with large cardinalities. Second, sq should be small to conserve memory since sq is the average number of sampled pairs. Last, qt cannot be very small to guarantee enough number of hosts with cardinalities close to t. As a rule of thumb, we find choosing s between 10% to 50%, and $q = O(1/t)$, say $0.5/t$ gives good performance.

IV. Experimental Results

In this section, we discuss the experimental results of applying our approach for identifying high cardinality hosts defined by various thresholds and cardinality ranges. We compare our method to two alternative methods using real Internet traffic

[8]There is a small positive bias due to the Bloom Filter collision, but this can be overcome by increasing the Bloom Filter memory.

Trace	# Source	# Pairs	> 50	> 100	> 1000
TR_1	6.14M	31.26M	34484	22154	2283
TR_2	14.26M	79.18M	56681	41854	5947
TR_3	1.35M	3.96M	2237	1422	116
TR_4	1.53M	4.91M	2401	1507	138

TABLE I
CHARACTERISTICS OF TRACES

traces. Experimental results show that our method outperforms the existing methods in both accuracy and memory usage.

A. Datasets

In order to evaluate the performance of our algorithm under real network environment, we choose two types of large-volume traces provided by a university network and an ISP, respectively. Table I shows the numbers of source hosts and pairs in those traces, along with the number of sources with cardinalities greater than 50, 100 and 1000, respectively[9]. All traces contain more than 1 million sources and a significantly large number of hosts with cardinalities above 50.

University Traces include a one-day flow trace (TR_1) and a one-week flow trace (TR_2). Both traces consist of *unsampled* traffic flows captured at the border router of a large campus network. Close examination of those traces reveals a significant amount of p2p and scanning traffic.

ISP Traces are provided by one of the major ISPs, captured at the Internet backbone. Two traces (TR_3 and TR_4) consist of *sampled* traffic flows from two separate days. The ISP traces are different from those university traces in that they contain much more variety of traffic types, such as traffic related to large commercial websites, or global malicious activities like botnets, worms and DDoS attacks, etc.

B. Characteristics of Cardinality Distributions

Even though these four traces come from two heterogeneous sources, they demonstrate high similarity in their cardinality distributions. Fig. 4 depicts the cumulative distribution of source cardinalities. We observe in Fig. 4 that hosts with small cardinalities dominate in all four traces. There are greater than 75% of hosts with cardinalities less than 3; greater than 97% of hosts with cardinalities less than 8. In addition, those cardinality distributions are heavy tailed. A noticeably large number of hosts are observed to have large or extremely large cardinalities (also see Table I).

Analysis on these traces reveals the causes of low cardinality hosts: 1) Limited number/location of vantage points. 2) Sampling effect during traffic trace collection. 3) Specific network activities like backscatters and p2p applications. In a word, traffic dominated by low cardinality hosts is a common property of Internet traffic, which makes our algorithm superior over other existing methods, since our algorithm is able to significantly reduce the number of low cardinality hosts to reserve resource for high cardinality host identification.

In the following subsections, we evaluate our approach from various perspectives. We first evaluate the K dependent sampling in Section IV-C. Then we discuss our experiment settings in Section IV-D. In Section IV-E, we compare our algorithm to two alternative algorithms for identifying high cardinality hosts defined by various threshold t. Since one unique property of our algorithm is its ability to identifiy hosts with cardinalities within predefined ranges, we study the range estimation results in Section IV-F. Finally, we discuss the bias correction results in detail in Section IV-G.

[9]All the traces are in Cisco NetFlow format. However, our method will also work on packet traces. In fact, due to more duplication of pairs within packet traces in general, our method is expected to achieve better results.

C. Evaluation of K dependent sampling

Before evaluating our algorithm as a whole, we first study the effectiveness of K dependent sampling method. The experimental result on trace TR_1 complements our theoretical analysis (Fig. 3) in Section II, and demonstrates the advantage of dependent sampling on real traffic traces.

Figure 5(a) displays the cumulative cardinality distribution of the candidates passing K parallel pair samplers with rate $r = 0.1$. The sampling rate r is chosen such that all hosts with cardinality beyond 50 will be selected with 99% probability(Eq. 2). It is clear from the plot that the percentage of candidate hosts with small cardinalities decreases significantly when K increases, and that dependent sampling works significantly better than its independent counterpart. Such observations are confirmed by Fig. 5(b), which shows the total number of candidates with cardinality N. Notice in Fig. 5(b) we have scaled the total number of hosts before filtering to 1 million. Interestingly, when $K = 3$, we can see that the curve flattens for small cardinalities. This combined with the fact that the candidates with somewhat larger cardinalities (more than 20) do not change much for different K, indicates that if we further increase K, the memory increase may outweigh the benefit of host reduction. Take TR_1 as an example, the number of candidates recorded by dependent sampling (number of hosts recorded by each of the K bloom filters), for $K = 1, \ldots, 4$ are: 5M, 1.01M, 0.62M, 0.48M, respectively, which translates to the overall host selection probabilities: $0.16, 0.034, 0.019, 0.015$. In comparison, the number of hosts recorded by independent sampling is: 5M, 1.45M, 0.75M, 0.55M for $K = 1, \ldots, 4$. The difference between $K = 1$ and $K = 2$ is certainly striking. Considering both memory usage and filtering efficiency, we fix $K = 3$ for all the subsequent experiments.

D. Experiment Settings

To highlight the ability of our algorithm for identifying hosts with moderately large cardinalities, we let threshold t to be 50, 100 and 1000, which are selected for detecting p2p hosts and scanners. To achieve a fair comparison among all the candidate algorithms, we choose the optimal configuration for each individual algorithm to attain the same *20% expected relative error rate*.

Table II shows the configurations of our algorithm given various threshold t, where r_0, r and f_1 denote the sampling rates for pre-filtering, parallel filtering and thresholded bitmap counting, respectively. Since many high cardinality hosts may have cardinalities much beyond the threshold t, a $E(L_K)$ value

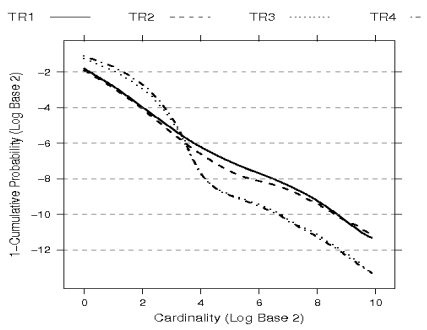
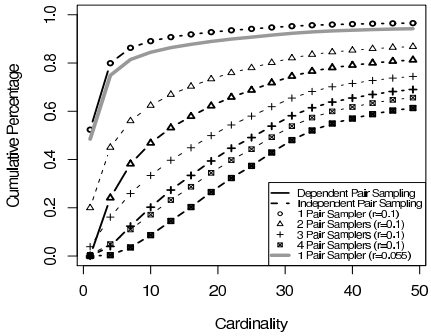
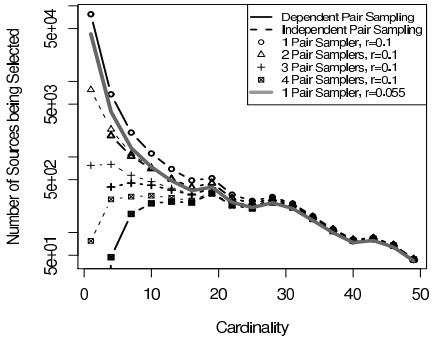


Fig. 4. Cumulative cardinality distributions.



(a) Cumulative cardinality distribution



(b) No. of candidates after filtering vs. cardinalities.

Fig. 5. Performance results of K ($K \leq 4$) parallel pair samplers.

t	r_0	r	f_1	s	q
50	0.150	0.200	0.040	0.100	0.010
100	0.075	0.120	0.020	0.100	0.010
1000	0.008	0.015	0.002	0.500	0.001

TABLE II
CONFIGURATIONS OF OUR ALGORITHM FOR DIFFERENT THRESHOLD t

(the expected number of pairs seen during two-phase filtering) between 20% or 30% of t may be good enough. We use the worst bound in Eq. (7) to choose r_0 and r such that both $E(L_K^{(0)})$ and $E(L_K^{(1)})$ is about half of $E(L_K)$. In addition, f_1 is determined by applying Eq. 9.

In addition, to estimate the number of missed pairs, when $t = 50$, we sample $s = 10\%$ of all the hosts and apply the sampling algorithm in [15] to obtain unbiased cardinality estimates for these 10% hosts. The pair sampling rate q is set to 1% and the Bloom Filter is configured using 3 independent hash functions. In addition to the cost of the new Bloom Filter, this bias correction method requires one more bit in each entry of the original hash table, which indicates whether the corresponding host is among these 10% sampled ones or not. When any of the 10% hosts starts to be sampled by the pair sampler, we append another 16 bits to the hash entry to count its cardinality. For $t = 100$ and $t = 1000$, s and q are set according to Table II. Notice these parameter settings depend on the cardinality distribution of the traffic trace. In practice, these parameters can be chosen using historical traffic data.

For comparison purpose, the first alternative algorithm that we select is the sampling approach introduced by [15]. This algorithm samples pairs with a rate r_p and store the sampled pairs in a Bloom Filter. Meanwhile, a hash table is used to count the cardinalities of those source hosts corresponding to the sampled pairs. We configure the Bloom Filter to use 3 independent hash functions as in our algorithm. In order to reduce the memory usage by the Bloom Filter, we stop recording all the remaining pairs associated with a specific host whenever its cardinality estimate equals or exceeds t. Hence, comparing to the original algorithm in [15], we decreased the number of pairs recorded and hence the memory used by the Bloom Filter. Based on the criteria of 20% relative error rate, we choose the sampling rate $r_p = 0.18$ for $t = 50$, $r_p = 0.10$ for $t = 100$ and $r_p = 0.015$ for $t = 1000$, respectively.

We choose the top-k cardinality estimating algorithm in [13] as the second alternative for comparison. There the approach is to randomly group sources into buckets, then estimate the source cardinality by the minimum cardinalities of those groups that contain the source. Take $TR1$ as an example. Since the average number of peers that each source communicates with is about 5, therefore, for a threshold $t = 50$, the expected number of sources that falls into each group has to be less than 10 in order for this to produce any meaningful result. This implies that the number of groups that we need to provision is at least $6M/10 = 600K$. This coupled with the fact that for each group we need at least 400 bits to reach a relative precision of 10%, meaning that the memory is at least 30M even in the best possible scenario. This is far too expensive compared to both our approach and [15]. During experiments, the second alternative algorithm also performs much worse then the others. Due to space limitation, we do not include the results for the second alternative algorithm in our paper.

E. Results for Identifying High Cardinality Hosts

In this section, we present the results of comparing our approach to the alternative algorithm in [15] for identifying hosts with cardinalities greater than t for $t = 50, 100, 1000$. We choose a university trace TR_1 and an ISP trace TR_3 as representatives for our experiments[10].

1) Accuracy

To evaluate the identification accuracy, we focus on the false positive rate (*f.p.r*) and false negative rate (*f.n.r*) of each algorithm. For each threshold t, we define positive instances (*p.i*) as the hosts whose real cardinalities are above t; while the false positives (*f.p*) refer to those hosts whose real cardinalities exceed t, but their estimated cardinalities are below t. The false negatives (*f.n*) and negative instances (*n.i*) can be defined in the similar way. The false positive rate and false negative rate are defined as: $f.p.r := |f.p|/|n.i|$ and $f.n.r := |f.n|/|p.i|$. The estimation results are shown in Table III. It is easy to see that about half of the time, results from our approach are significantly better than the alternative algorithm in [15], and for the other half, results are comparable.

2) Memory Usage

The memory consumption of our algorithm consists of two parts: the $K + 1$ Bloom Filters and the hash table for thresholded bitmap counting. The memory used by $K + 1$ Bloom

[10]Since TR_3 only contains few hosts with cardinality above 1K, we exclude them from the experiment for $t = 1000$. For the same reason, we do not consider TR_3 for estimating hosts within ranges of [1K, 5K] and [5K, 25K].

	Our Algorithm				Alternative Algorithm			
t	*f.p.r* (%)		*f.n.r* (%)		*f.p.r* (%)		*f.n.r* (%)	
	TR_1	TR_3	TR_1	TR_3	TR_1	TR_3	TR_1	TR_3
50	0.02	0.08	5.60	6.53	0.07	0.01	5.31	5.45
100	0.02	0.01	3.88	2.85	0.04	0.01	6.30	8.72
1000	0.002	NA	5.7	NA	0.04	NA	6.31	NA

TABLE III
ESTIMATION RESULTS FOR OUR ALGORITHM TO THE ALTERNATIVE ALGORITHM BY KAMIYAMA ET.AL

Filters are determined by the total number of source hosts recorded by those Bloom Filters. Since all Bloom Filters are configured using 3 independent hash functions with $2^{-3} = 0.125$ expected collision probability, from [3], we need to assign $3/\ln 2 = 4.32$ bits per recorded host. The total memory consumed by the two-phase filtering is $4.32(H_0 + KH_0')$, where H_0 and H_0' stand for the number of hosts recorded by the pre-filtering step and the parallel filtering step, respectively. During the counting stage, we apply a chained hash table for storing the thresholded bitmap counters. The memory required by the hash table can be estimated by counting the total number of entries in the hash table. In our experiment, we use a virtual bitmap of 40 bits for counting the cardinality of each individual host. From [9], 40 bits corresponds to a relative accuracy of about 20% at the threshold t. Therefore, each entry in the hash table consumes 32 bits for storing the host address (IP address) as a key, 40 bits for the virtual bitmap counter and $\log_2 H$ bits for the pointer connecting the key to the counter, where H stands for the total number of entries (candidates) in the hash table. Therefore, the memory usage by the counting process is $(72 + \log_2 H)H$. The bias correction consumes a small extra memory comparing to the filtering and counting process, which we will discuss separately in Section IV-G.

The memory usage of the algorithm in [15] also consists of a Bloom Filter for pair sampling with rate r_p and a hash table for cardinality estimation. However, the Bloom Filter in the alternative algorithm records pairs instead of source hosts, hence the total memory usage for the Bloom Filter is 4.32 bits per recorded pair. Obviously, with the same collison rate, the Bloom Filter for recording pair requires far more memory than the Bloom Filter for storing hosts as in our approach, since the number of pairs is in general 5 times more than the number of hosts. To count the memory requirement for the hash table, each entry in the hash table needs 32 bits for the host address as a key, an average of $\log_2(t \cdot r_p)$ bits for cardinality counting, and $\log_2(H)$ bits for the pointer. The total memory used by the hash table is $(32 + \log_2(t \cdot r_p) + \log(H))H$.

Table IV presents comparisons between our approach and [15] in terms of the number of finally selected hosts (with non-zero cardinality estimates) and total memory consumption. A key observation is that, for applications of identifying high cardinality hosts, with equivalent or even better accuracy, we achieve on average a factor of 12 reduction in the number of candidate hosts, and a factor of 4.5 reduction in memory usage, regardless of traces and thresholds. Clearly, our algorithm outperforms the alternative algorithm, since we have significantly reduced the low cardinality hosts, thus we can devote more resource to each candidate to obtain a more accurate estimate.

t	Trace	Our Algorithm		Alternative Algorithm	
		Memory (KB)	Number of Hosts (10^3)	Memory (KB)	Number of Hosts (10^3)
50	TR_1	2840	139	12992	1640
100	TR_1	1679	85	7917	990
1000	TR_1	278	14	1517	187
50	TR_3	776	35	3321	415
100	TR_3	324	11	2013	253

TABLE IV
COMPARISON OF MEMORY USAGE AND NUMBER OF CANDIDATE HOSTS

Range	Missed SD Pairs		*f.p.r* %		*f.n.r* %	
	TR_1	TR_3	TR_1	TR_3	TR_1	TR_3
50-250	16.68	25.89	0.12	0.06	3.63	1.3
250-1250	87.68	120.85	0.04	0.01	1.59	0.7
100-500	36.87	52.39	0.08	0.03	3.00	2.0
500-2500	97.39	141.17	0.01	0.002	4.44	5.0
1K-5K	241.52	NA	0.006	NA	4.44	NA
5K-25K	1501.38	NA	0.001	NA	5.29	NA

TABLE V
CARDINALITY RANGE ESTIMATION RESULTS

F. Results of Range Cardinality Estimation

One major advantage of our algorithm is its ability to identify the hosts with cardinalities within specific ranges, using the thresholded bitmap algorithm presented in Section III-A which only requires a negligible amount of extra memory comparing to a single bitmap. Our algorithm for range estimation works as follows: we first partition all possible cardinalities into I ranges, like $[t, 5t)$, $[5t, 5^2t)$ and so on. We choose I sampling rates for the virtual map counting corresponding to those ranges using Eq. 9. The cardinality estimates from (10) are corrected using an estimate of the average number of missed pairs discussed in Section III-B. The remaining components are configured according to Table II. The estimation results are summarized in Table V. We can see that our algorithm has the capability of accurately identifying high cardinality hosts within specific ranges, with all false negative rates below 5% and negligible false positive rates.

We measure the estimation error of individual hosts using the *mean square relative error*, which is defined as:

$$\sqrt{\frac{1}{|\mathcal{H}|}\sum_{h\in\mathcal{H}}\left(\frac{\hat{N}_h - N_h}{N_h}\right)^2}$$

where $\hat{N}_h$ and N_h denote the estimated cardinality and real cardinality of host h, respectively, and $\mathcal{H}$ stands for the set of hosts whose cardinalities are within the predefined range. The experimental result shows that regardless of the chosen range, our algorithm always gets a mean square relative error around 18%, consistent with our 20% design goal.

G. Bias Correction with the Average Number of Missed Pairs

Another contribution of our paper is the bias correction method for estimating the number of missed pairs in the filtering and counting process. In this section, we study the significance of bias correction in our estimation results.

Table VI depicts the number of false positives (*f.p*) and false negatives (*f.n*) before and after we apply bias correction on trace TR_1. We observe that before the bias correction procedure, the estimation results tend to have more false

		Before Correction			After Correction		
t	Memory	*f.p*	*f.n*	*f.p.r*	*f.p*	*f.n*	*f.p.r*
50	75KB	958	2848	0.083	1708	1932	0.056
100	59KB	468	2095	0.095	1457	860	0.039

TABLE VI
EFFECT OF BIAS CORRECTION

negatives. It means that we underestimate the cardinalities of those candidates due to the missed information at the filtering stage. However, after bias correction we obtain a more even distribution of false positives and false negatives. The false negative rates change from 10% to around 5% in general[11].

Moreover, for $t = 50$, we observe a 4.4% decrease of the number of total errors (the number of false positives plus the number of false negatives) after applying bias correction. For $t = 100$, the decrease of total errors reaches 9.6%. We conclude that with a small amount of additional memory (1% - 3%, see Table IV and Table VI), the bias correction process help reduce a significant portion of the total errors.

V. OTHER RELATED WORK

There are a large body of work on sampling methods related to network traffic analysis [5], [7], [6], [10]. In a streaming context, many effective stream sampling methods have been developed for estimating specific aggregates such as quantiles [11], heavy hitters [16], distinct counts [10], subset-sums [7], set resemblance and rarity [4] etc., as well as generic sampling such as fixed-size reservoir sampling [19], adaptive geometric sampling [2], [12], etc. Implementation of these stream sampling techniques has been addressed in [14].

VI. CONCLUSIONS

In this paper, we have developed a sampling approach to identify high cardinality hosts defined by a moderately large threshold t or predefined ranges. We proposed a two-phase filtering approach which significantly reduce the number of low cardinality hosts. We also designed a thresholded bitmap method for estimating host cardinalities. In addition, we developed a bias correction method to compensate for the bias caused by the filtering and counting process. The experimental results on real Internet traces indicated that our algorithm noticeably reduced the memory consumption while achieving better accuracy comparing to alternative algorithms.

ACKNOWLEDGMENTS

The work is supported in part by the National Science Foundation grants CNS-0626808, CNS-0626812 and CRI 0709048.

APPENDIX

Proof of Theorem 1.

Proof: For a host with cardinality N, and for K parallel samplers each with a sampling rate r, let $\{I_1, \ldots, I_K\}$ represent the filtering outcome of the host: $I_i = 1$ means at least one of its pair is sampled by the ith sampler, and $I_i = 0$ otherwise. Let $P(\{I_1, \ldots, I_K\})$ denote the probability of this filtering outcome (for simplicity, we omit the dependency of P on N, r). It is easy to show that

$$P(\{0, \ldots, 0\}) = (1 - Kr)^N. \quad (12)$$

[11]The false positive rates are too small due to the large denominator (the number of negative instances) and hence are not listed in Table VI.

In the following, we shall show how $P(\{1, \ldots, 1\})$ (which is same as $p_K(N)$ in Theorem 1) can be derived from manipulations with respect to $P(\{0, \ldots, 0\})$ in (12). We illustrate the proof for $K = 2$ and $K = 3$. The general case can be shown via similar induction.

With the pair sampling rate r, from (12),

$$P(\{0, 0\}) = (1 - 2r)^N, \quad P(\{0\}) = (1 - r)^N.$$

Now with $K = 2$ parallel pair samplers, there are four outcomes for a host with cardinality N: $\{0, 0\}, \{0, 1\}, \{1, 0\}, \{1, 1\}$. It is obvious that:

$$P(\{0, 1\}) = P(\{1, 0\}) = P(\{0\}) - P(\{0, 0\}) = (1-r)^N - (1-2r)^N$$

and thus

$$\begin{aligned} P(\{1, 1\}) &= P(\{1\}) - P(\{0, 1\}) = 1 - P(\{0\}) - P(\{0, 1\}) \\ &= 1 - 2P(\{0\}) - P(\{0, 0\}) = 1 - 2(1 - r)^N + (1 - 2r)^N. \end{aligned}$$

When $K = 3$, there are 8 outcomes. From (12), $P(\{0, 0, 0\}) = (1 - 3r)^N$. It is now easy to see that

$$\begin{aligned} P(\{1, 1, 1\}) &= P(\{1, 1\}) - P(\{0, 1, 1\}) \\ &= 1 - 2P(\{0\}) - P(\{0, 0\}) \\ &\quad -(P(\{0\} - 2P(\{0, 0\}) - P(\{0, 0, 0\})) \\ &= 1 - (1 - 3r)^N + 3(1 - 2r)^N - 3(1 - r)^N. \end{aligned}$$

■

REFERENCES

[1] Identifying High Cardinality Internet Hosts. Technical report, 2008.
[2] A. Bagchi, A. Chaudhary, D. Eppstein, and M. T. Goodrich. Deterministic sampling and range counting in geometric data streams. *ACM Trans. Algorithms*, 3(2):16, 2007.
[3] B. Bloom. Space/time trade-offs in hashing coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
[4] M. Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. In *ESA '02*, pages 323–334, 2002.
[5] N. Duffield, C. Lund, and M. Thorup. Flow sampling under hard resource constraints. In *SIGMETRICS '04/Performance '04*, 2004.
[6] N. G. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. *IEEE/ACM Trans. Netw.*, 13(5), 2005.
[7] N. G. Duffield, C. Lund, and M. Thorup. Learn more, sample less: control of volume and variance in network measurement. *IEEE Transactions on Information Theory*, 51(5):1756–1775, 2005.
[8] M. Durand and P. Flajolet. Loglog counting of large cardinalities. In *Proceedings of European Symposium on Algorithms*, 2003.
[9] C. Estan, G. Varghese, and M. Fisk. Bitmap algorithms for counting active flows on high speed links. *IEEE/ACM Trans. on Networking*, 14(5):925–937, 2006.
[10] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB '01*, pages 541–550, 2001.
[11] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of SIGMOD 2001*, 2001.
[12] J. Hershberger and S. Suri. Adaptive sampling for geometric problems over data streams. In *PODS '04*, pages 252–262, 2004.
[13] K. Ishibashi, T. Mori, R. Kawahara, Y. Hirokawa, A. Kobayashi, and K. Y. H. Sakamoto. Estimating top n hosts in cardinality using small memory resources. In *ICDEW '06*, page 29, 2006.
[14] T. Johnson, S. Muthukrishnan, and I. Rozenbaum. Sampling algorithms in a stream operator. In *SIGMOD '05*, pages 1–12, 2005.
[15] N. Kamiyama, T. Mori, and R. Kawahara. Simple and adaptive identification of superspreaders by flow sampling. In *INFOCOM 2007*, pages 2481–2485, 2007.
[16] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB '02*, pages 346–357, 2002.
[17] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *OSDI'04*, 2004.
[18] S. Venkataraman, D. X. Song, P. B. Gibbons, and A. Blum. New streaming algorithms for fast detection of superspreaders. In *NDSS 2005*.
[19] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.