Hierarchical Virtual Bitmaps for Spread Estimation in Traffic Measurement

Olufemi Odegbile[1], Chaoyi Ma[2], Shigang Chen[3], Dimitrios Melissourgos[4] and Haibo Wang[5]

Department of Computer and Information Science and Engineering
University of Florida, Gainesville, Florida, USA
Email: {[1]oodegbile, [2]ch.ma, [3]sgchen, [4]dmelissourgos, and [5]wanghaibo}@ufl.edu

Abstract. This paper introduces a hierarchical traffic model for spread measurement of network traffic flows. The hierarchical model, which aggregates lower level flows into higher-level flows in a hierarchical structure, will allow us to measure network traffic at different granularities at once to support diverse traffic analysis from a grand view to fine-grained details. The spread of a flow is the number of distinct elements (under measurement) in the flow, where the flow label (that identifies packets belonging to the flow) and the elements (which are defined based on application need) can be found in packet headers or payload. Traditional flow spread estimators are designed without hierarchical traffic modeling in mind, and incur high overhead when they are applied to each level of the traffic hierarchy. In this paper, we propose a new Hierarchical Virtual bitmap Estimator (HVE) that performs simultaneous multi-level traffic measurement, at the same cost of a traditional estimator, without degrading measurement accuracy. We implement the proposed solution and perform experiments based on real traffic traces. The experimental results demonstrate that HVE improves measurement throughput by 43% to 155%, thanks to the reduction of per-packet processing overhead. For small to medium flows, its measurement accuracy is largely similar to traditional estimators that work at one level at a time. For large aggregate and base flows, its accuracy is better, with up to 97% smaller error in our experiments.

1 Introduction

Traffic measurement is critical in supporting modern network functions [1], [2], [3], [4], [5], [6],[7]. Accurate information about current traffic loads is needed for routing optimization and load balancing among middleboxes that provide web proxing, firewalling and other functions [4], [5]. Network statistics are widely used to establish normal traffic patterns and detect anomalies that deviate from the normal [8]. Flow-level measurement provides fine-grained data to assess the behavior of individual hosts or subnets for performance or cybersecurity analysis [9]. While packet forwarding is the key function for any high-speed switch or router, auxiliary functions such as traffic measurement should be made as space-time efficient as possible, not only to avoid becoming a throughput bottleneck but also to save resources (e.g., cache memory and hardware circuits) for other important functions.

NetFlow [10] is a commonly used traffic measurement tool. It provides statistics such as number of packets and number of bytes for each TCP flow. A flow is a set of packets identified by common user-defined characteristics, such as source and destination IP addresses, source and destination ports, and protocol types. This paper extends the measurement function in two ways. First, we consider a hierarchical flow model, which we explain through an example where a cloud provider allocates vitual machines (VMs), racks of physical machines or whole pods to its clients, where each pod contains multiple racks. Suppose the provider wants to implement a measurement function at its datacenter gateway to monitor traffic between its clients and the Internet. Flows can be defined at the level of VMs, where each *VM flow* consists of all packets from a VM to the Internet (or from the Internet to a VM). They can also be defined at the rack level, where each *rack flow* consists of all packets from a rack to the Internet or vice versa. They can even be defined at the pod level, where a *pod flow* consists of all packets from a pod to the Internet or vice versa. These flows are organized in a three-level hierarchy, where each pod flow contains multiple rack flows, each rack flow consists of multiple VM flows, and each packet belongs to one flow at each level. Measuring the flow spread, which is the number of distinct elements (under measurement) in a flow, at different levels provide information with different granularities for traffic analysis.

Second, instead of the simple metric of packet number, we can measure any *elements* that are defined according to an application's requirements and carried in the packet headers or payload. Use the previous cloud example. The provider may monitor its clients' outbound traffic for suspicious activities, where each VM (rack or pod) flow consists of all packets from the VM (rack or pod) to the Internet. In particular, it may measure the number of distinct destinations in each flow. For instance, if a VM flow contacts too many destination addresses than it normally does, the VM may be used as a bot for scanning. If a rack or pod flow contains too many destination addresses than normal, even though its individual VMs appear to behave within bound, the overall aggressive behavior at the pod level may signal a botnet activity or a worm activity with many VMs in the pod compromised for *stealthy* worm propagation, where each infected host restrains its scanning rate from being too high to avoid detection.

Summarizing the above discussions, the flow model in a typical datacenter is an hierarchical model. Although hierarchical models are usually applied in a number of multi-layer networks this paper explores the model to reduce traffic measurement overheads in spread estimation. To this end, we proposed a new form of traffic measurement, called *hierarchical spread estimation*, which is not studied before. It estimates flows' spreads, where all flows are organized in a hierarchical structure, offering different granularites in traffic measurement at once.

Measuring flow spread at a single level has been studied before with two classes of solutions: One class measures each flow separately, keeping track of all flow labels and assigning a separate data structure for each flow to encode its elements [11], [12], [13]; the other class is more memory-efficient by encoding the elements from all flows in a single compact data structure without keeping track of flow labels — given a flow label (obtained by other means or of interest to the admin), the compact data structure can estimate the flow spread [14],[15]. Due to its excellent space-time efficiency, this paper will focus on the second class by extending the problem from a flat single level of flows (such as TCP flows traditionally) to a generalized multi-level hierarchy of flows. One may argue that the traditional single-level solutions can be simply applied to every level of a hierarchy. The problem is that each arrival packet will need to be processed multiple times, one at every level. [16] propose a solution that uses counter to track sizes (number of packets) of hierarchical flows. However, this method is unsuitable for spread estimation because counters cannot keep track of addresses in order to remove duplicates.

The contribution of this paper is to conduct the first study on hierarchical spread estimation, with an efficient solution that physically processes each arrival packet only once, while logically encoding the element of the packet at all levels for all relevant flows. It achieves the benefits of multi-grained traffic measurement at the same cost of traditional single-level solutions. Technically, we propose a new hierarchical virtual bitmap architecture, which shares bits not only among flows at the same level to save cache memory, but also among flows across levels such that encoding an element from an arrival packet at the lowest level of the hierarchy will automatically propagate the encoded information through all levels, regardless of the number of levels there are. We mathematically derive the formula of our hierarchical spread estimator, and prove that the proposed estimator is asymptotically unbiased. We implement the estimator both in software and hardware. Through extensive experiments, we demonstrate that compared to the state-of-the-art, hierarchical virtual bitmap estimator (HVE) delivers from 43% to 155% more throughput while its accuracy is in general close and even has up to 97% smaller error for some large upper-level flows. The rest of the paper is organized as follows. Section 2 presents the flow model, system model and formulates our research problem. Section 3 discusses the related works. We present the detailed architecture of our solution (HVE) in Section 4. Section 5 extensively evaluates HVE. Section 6 concludes the paper.

2 Flow Model, System Model and Problem Statement

2.1 Flow Model

Consider a hierarchical flow model of l levels. Flows at each level are disjoint, while a flow f_{j-1} at the $(j-1)$th level contains multiple child flows f_jth at the jth level, forming a hierarchical structure among flows of different levels, as illustrated in Fig. 1, where $1 < j \leq l$.

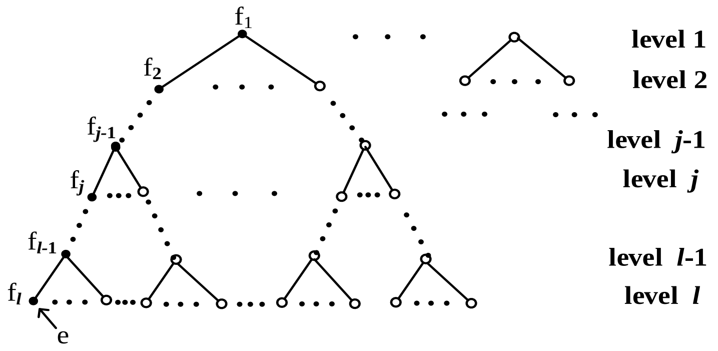

Fig. 1: An example of a hierarchical flow model of l levels.

Each packet belongs to a single flow f_l at the lth level. It also belongs to the flow parent at the $(l-1)$th level, to that flow parent at the $(l-2)$th level, ..., and all the way to a flow at the first level f_1, which contains f_l as a descendant in the hierarchy. Flows at the lth level are called *base flows*; flows at other levels are called *aggregate flows.*

2.2 System Model

Our system consists of gateway routers/switches responsible for traffic measurement. When a packet arrives at a switch/router, the labels of the base/aggregate flows that it belongs to (such as $f_1, \ldots, f_l$) are extracted from the packet header. For example, suppose the source address is the base flow label at the first level, the address 24-bit prefix is the second-level flow label, and the address 16-bit prefix is the third-level flow label. The switch can easily obtain all flow labels by extracting the source address from the packet header and applying appropriate bit masks. After obtaining f_1 through f_l, the switch also extracts the element e, which are defined based on application need and can be found in packet headers or payload from the packet. In case that the destination address is the element, the switch will copy it from the header.

2.3 Problem Statement

Given an arrival packet stream at a switch/router, the problem is to measure the spreads of all flows in the hierarchy, including both base flows and aggregate flows. Our goal is to design a hierarchical spread estimator that encodes each packet only once overall, instead of once for each of l base/aggregate flows it belongs to, yet being able to provide accurate spread estimation for all flows. The design of such an estimator includes two operations:

- Online element encoding: It stores distinct elements from all flows in a compact data structure for online operation at the same place where packet forwarding is performed
- Offline Spread estimation: It takes the encoded data and calculates an estimation for the spread of any given flow.

3 Preliminaries

In this section, we briefly review related methods of estimating a flow spread. Suppose that an incoming flow at a switch w is represented as $\langle f, e \rangle$, where f is a flow id and e is an element id. The spread of f is the number of distinct elements encoded during a measurement period. Let n be the actual spread of f. In what follows, we will discuss how n can be estimated through the methods mentioned above.

3.1 Bitmaps

[11] propose the bitmap as a lightweight and compact data structure to estimate the spread of a flow. In order to estimate n with a bitmap, an array B, which contains m bits initialized to zeroes, is allocated to store distinct elements of f. For each element e, we randomly set a location k^* in B as 1, that is

$$B[k^*] = 1 \tag{1}$$

where $k^* = H(e) \bmod m$ and $H(\cdot)$ is a hash function. Once all contacts are stored in B, n is estimated as

$$\hat{n} = -m \ln V_m \tag{2}$$

where V_m is the fraction of bits in B that are still '0' at the end of the measurement period.

3.2 Opensketch(Bitmap)

A bitmap is ideal for estimating the spread of one flow. If we have to estimate the spreads of multiple flows, then we will need to construct an independent bitmap for each flow. Consequently, the combined size of independent bitmaps is proportional to the number of flows monitored in a measurement period. More compact and memory efficient spread estimators are based on an idea of memory sharing, where all flows' elements are encoded in a shared physical memory.

A memory sharing spread estimator based on CountMin [17] is proposed by OpenSketch [3], which replaces counters in CountMin with uniform bitmaps. We will refer to this estimator as *opensketch(bitmap)*. Let B be an array of k bitmaps.

For each data item $\langle f, e \rangle$, it hashes f to k bitmaps, then hashes e to a bit in each bitmap, and sets that bit to one,

$$B[H_i(f)][H(e)] = 1, 0 \leq i < k. \tag{3}$$

Each of the k bitmaps for f produces an estimate for the flow spread, which carries noise from other flows due to hash collision. Final estimate for the spread of f is the minimum estimate from the k bitmaps, since it contains the least noise.

3.3 Virtual Bitmap

Instead of constructing independent bitmaps, [14] randomly construct *virtual bitmap-s* from a shared pool of physical array of bits. Let P be a shared array of m bits. Let the size of each virtual bitmap be s. The virtual bitmap of f, denoted as X_f, is generated in the following way:

$$X_f[i] = P[H_i(f)],\ 0 \leq i < s, \tag{4}$$

where H_i, $0 \leq i < s$, are independent hash functions. For each arrival element e of f, we randomly select a location k^* in X_f as

$$k^* = H(e) \bmod s, \tag{5}$$

where $H(\cdot)$ is a hash function. Next, we set the bit at location k^* in X_f to 1:

$$X_f[k^*] = P[H_{k^*}(f)] = 1. \tag{6}$$

This implies that, through virtual bitmaps, all distinct elements belonging to f and all others flows are stored in P. Unfortunately, X_f not only contains the spread of f, but potentially contains some noise from other flows. Hence, the estimated value of n is

$$\hat{n} = s\ln(V_m) - s\ln(V_s), \tag{7}$$

where V_s and V_m are the fractions of bits that are still '0' in X_f and P, respectively, at the end of a measurement period. The first term on the right hand side of equation (7) is the estimated noise contained in X_f.

3.4 Other Related Works

Our focus in this paper is on estimation of flow spread. Past solutions use hash-based compact data structures called *sketches* [11], [12], [13], [18], [14], [15], [19], [20], [21], [3], [22], [23], [24]. These solutions can be divided into three categories. The first category compactly estimates the spread of a single flow [11], [12], [13], [18], [25]. As a result, the total memory allocation is proportional to the number of flows being monitored. The second category estimates the spreads of multiple flows by using a shared pool of resources, either bits or counters [14], [15], [19], [20], [21]. This is done by constructing virtual sketches from the shared memory. The third category introduces universal and adaptive sketches. This category not only estimates flow spread, but can also be used to perform other tasks such as identification of heavy hitters and detection of traffic changes [3], [22], [23], [24], [25].

4 Hierarchical Virtual Bitmap Estimator (HVE)

There are two main operations in our Hierarchical Virtual bitmap Estimator (HVE). The first operation deals with online data encoding. The second operation is spread estimation, which is carried out either by the control plane of the switch/router that performs online encoding or by a centralized controller.

4.1 Virtual Arrays

The data structure for HVE is simply an array B of m bits, which are initialized to zeros at the beginning of each measurement period. A bit in the array is denoted as $B[k]$, $0 \leq k < m$. Our approach is to pseudo-randomly allocate a virtual array of bits from B to each base/aggregate flow to encode its elements.

Without lose of generality, consider an arbitrary packet, carrying an element e and belonging to a base flow f_l, whose parent chain is f_1, ..., f_{l-1}. Fig. 2 illustrates how bits are allocated for the virtual arrays of f_l through f_1, which are denoted as B_{f_j}, $1 \leq j \leq l$. Because flows are dependent, we do not independently assign physical bits to flows. Rather, each flow takes some bits from its parent's virtual array to form its own virtual array. Since first-level flows have no parents, they pseudo-randomly select bits from the physical array B.

Let $s_j, 1 \leq j \leq l$ be the pre-defined length for the virtual array of any flow at the jth level, where $s_j < s_{j-1}$, $1 < j \leq l$. We first describe how an arbitrary base flow f_l will select s_l bits for its virtual array, and then describe how an arbitrary jth-level aggregate flow f_j will select s_j bits from its parent f_{j-1}'s virtual array, $1 < j \leq l$. The bits in B_{f_1} are pseudo-randomly selected from B as follows in equation 8.

$$B_{f_1}[k] = B[H_k(f_1)],\ 0 \leq k < s_1, \tag{8}$$

where H_k, $0 \leq k < s_1$, is an independent hash function. We can replace the s_1 hash functions in (8) with a single master hash function H_M as follows:

$$H_k(f_1) = H_M(f_1 \oplus R_1[k]),\ 0 \leq k < s_1, \tag{9}$$

where R_1 is a set of s_1 random numbers and $\oplus$ is the XOR operator. By substituting (9) into (8), we have

$$B_{f_1}[k] = B[H_M(f_1 \oplus R_1[k])],\ 0 \leq k < s_1. \tag{10}$$

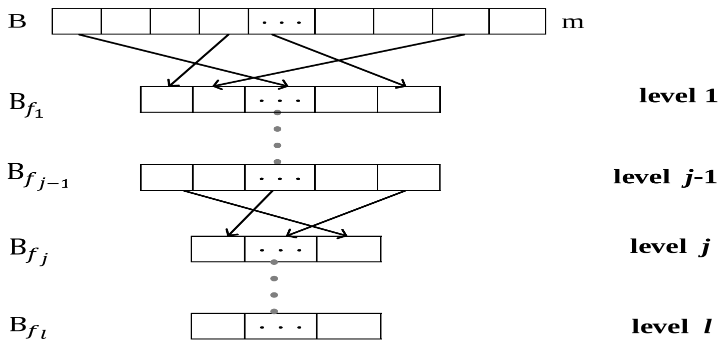

Fig. 2: Bits allocation in hierarchical virtual bitmaps.

Generalizing in equation (11), the bits in B_{f_j} are pseudo-randomly selected from $B_{f_{j-1}}$, where B_{f_j} is the virtual array of flow f_j at the jth level and $B_{f_{j-1}}$ is the virtual array of the parent flow f_{j-1} at the $(j-1)$th level.

$$B_{f_j}[k] = B_{f_{j-1}}[H_M(f_j \oplus R_j[k])],\ 0 \leq k < s_j. \tag{11}$$

These are virtual constructions that are not actually carried out online.

4.2 Online Encoding

A switch that receives an arrival packet, which belongs to f_1,...,f_l, will pseudo-randomly select a bit from the virtual array B_{f_l} of the highest-level flow f_l and encode the element by setting the bit to one. Recall that this bit is taken from its parent's virtual array $B_{f_{l-1}}$. Hence, by setting the bit, we have also encoded the element for the parent flow. As this argument repeats, by setting just one bit, we have actually encoded the elements for all flows f_l through f_1 in their virtual arrays.

However, we cannot operate directly on the virtual arrays, which are virtual after all. The bit we are setting is a physical bit, which is taken in the virtual arrays. Below we show how to select a bit from B_{f_l} to set and how this bit is translated into a physical bit in B for setting. In equation (12), the selection of a bit is done by hashing the element e for an index.

$$k^* = H_M(e) \bmod s_l \tag{12}$$

From equation (11), the virtual bit of B_{f_l} at index k^* is the following bit in at the $(l-1)$th level as in equation (13):

$$B_{f_l}[k^*] = B_{f_{l-1}}[H_M(f_l \oplus R_l[k^*])], \tag{13}$$

which is in turn the following bit at the $(l-2)$th level as in equation (14):

$$B_{f_{l-1}}[H_M(f_l \oplus R_l[k^*])] = B_{f_{l-2}}[H_M(f_{l-1} \oplus R_{l-1}[H_M(f_l \oplus R_l[k^*])])]. \tag{14}$$

Repeating the above process, we eventually reach a bit in the physical array B as in equation (15):

$$B_{f_l}[k^*] = B[H_M(f_1 \oplus R_1[H_M(f_2 \oplus R_2[\cdots H_M(f_l \oplus R_l[k^*])])])] \tag{15}$$

The only encoding action taken by the switch after receiving a packet is to set the above physical bit to one, as done in equation (16),

$$B[H_M(f_1 \oplus R_1[H_M(f_2 \oplus R_2[\cdots H_M(f_l \oplus R_l[k^*])])])] = 1 \tag{16}$$

This assignment automatically encodes the element in all l virtual arrays, B_{f_1}, B_{f_2}, $\cdots B_{f_l}$, for the flows that the packet belongs to, with $l+1$ hashes and one memory access. These hash computations can be pipelined in hardware implementation [16], which is very efficient as we observe in our GPU implementation. The full pipeline implementation encodes each packet in one clock cycle.

4.3 Spread Estimation

At the end of each measurement period, the physical array B is offloaded to the switch's control plane or to a centralized controller, where spread estimation is performed. Given an arbitrary flow label f_j at an arbitrary level $1 \leq j \leq l$, we first derive the labels on its parent chain, f_{l-1} through f_1. We then construct its virtual array B_{f_j} by copying its s_j bits from the physical array B as in equation (17):

$$B[H_M(f_1 \oplus R_1[H_M(f_2 \oplus R_2[\cdots H_M(f_j \oplus R_j[k])])])], 0 \leq k < s_j, \tag{17}$$

where f_1 is the parent flow to f_2, which is in turn the parent flow to f_3, and all the way to f_j. We stress that this construction happens during offline spread estimation, whereas no virtual array is constructed during the online operation of element encoding. We similarly construct the virtual arrays of $B_{f_{j-1}}$ through B_{f_1}. Our proposed spread estimator, HVE, is derived in Theorem 1:

Theorem 1. *Let n be the total number of distinct elements from all flows and n_{f_i} be the actual spread of flow f_i, where $1 \leq i \leq j$. Let U_m be the number of '0' bits in B and U_{f_i} be the number of '0' bits in the virtual array B_{f_i}. We define V_{f_i} as*

$$V_{f_0} = \frac{U_m}{m} \text{ and } V_{f_i} = \frac{U_{f_i}}{s_i}, 1 \leq i \leq j. \tag{18}$$

Then HVE *for n_{f_i} is*

$$\hat{n}_{f_j} \simeq s_j \ln(V_{f_{j-1}}) - s_j \ln(V_{f_j}), \tag{19}$$

when $1 < j \leq l$ *and*

$$\hat{n}_{f_1} \simeq s_1 \ln(V_{f_0}) - s_1 \ln(V_{f_1}), \tag{20}$$

when $j = 1$.

Proof. Before we derive our estimator, we first estimate the $\ln(E(V_{f_j}))$ in equation (21):

$$\ln(E(V_{f_j})) = -\frac{n}{m} - \frac{n_{f_1}}{s_1} - \frac{n_{f_2}}{s_2} - \cdots - \frac{n_{f_j}}{s_j}. \tag{21}$$

Let $A(f_j, k)$ be an event that the bit at an arbitrary index k in the virtual array B_{f_j} remains '0' at the end of a measurement period. Let $I(f_j, k)$ be a binary indicator, which is 1 if $A(f_j, k)$ happens, or 0 otherwise. Let $\hat{k}$ be the index of the physical bit in B that is selected for the bit at index k in B_{f_j}. Event $A(f_j, k)$ occurs *if, and only if,* none of the arrival packets sets the bit at index k in B to 1.

Let b be the bit at index k in B_{f_j}. Each of the n_{f_j} elements from flow f_j has a probability of $\frac{1}{s_j}$ to set the bit b as 1; each of the $n_{f_{j-1}} - n_{f_j}$ elements in its parent flow but not in f_j has a probability of $\frac{1}{s_{j-1}}$ to set the bit b as 1.

Continuing this line of reasoning, each of the $n - n_{f_1}$ elements not in f_1 has a probability of $\frac{1}{m}$ to set the bit b in B. Hence, the probability for $A(f_j, k)$ to happen is stated in equation (22):

$$Prob(A(f_j, k)) = \left(1 - \frac{1}{m}\right)^{n-n_{f_1}} \left(1 - \frac{1}{s_1}\right)^{n_{f_1}-n_{f_2}} \cdots \left(1 - \frac{1}{s_j}\right)^{n_{f_j}}, \; 0 \leq k \leq s_j - 1. \tag{22}$$

By definition, $U_{f_j} = \sum_{k=0}^{s_j-1} I(f_j, k)$. Therefore,

$$\begin{aligned} E(V_{f_j}) &= \frac{1}{s_j} \sum_{k=0}^{s_j-1} E(I(f_j, k)) \\ &= \frac{1}{s_j} \sum_{k=0}^{s_j-1} Prob(A(f_j, k)) \\ &= \left(1 - \frac{1}{m}\right)^{n-n_{f_1}} \cdots \left(1 - \frac{1}{s_j}\right)^{n_{f_j}} \end{aligned} \tag{23}$$

$E(V_{f_j})$ in (23) can be approximated as in equation (24)

$$E(V_{f_j}) \simeq e^{-\frac{n-n_{f_1}}{m}} e^{-\frac{n_{f_1}-n_{f_2}}{s_1}} \cdots e^{-\frac{n_{f_{j-1}}-n_{f_j}}{s_{j-1}}} e^{-\frac{n_{f_j}}{s_j}}, \tag{24}$$

when m, s_1, ..., s_j, $n - n_{f_1}$, $n_{f_1} - n_{f_2}$, ..., n_{f_j} are sufficiently large. Assume the spread of a child flow is much smaller than the spread of a parent flow (which contains many child flows), i.e., $n_{f_1} \ll n$, $n_{f_2} \ll n_1$, ..., $n_{f_j} \ll n_{j-1}$. We have

$$E(V_{f_j}) \simeq e^{-\frac{n}{m}-\frac{n_{f_1}}{s_1}\cdots-\frac{n_{f_{j-1}}}{s_{j-1}}-\frac{n_{f_j}}{s_j}}. \tag{25}$$

We rewrite equation (25) as equation (26)

$$\ln(E(V_{f_j})) \simeq -\frac{n}{m} - \frac{n_{f_1}}{s_1} \cdots - \frac{n_{f_{j-1}}}{s_{j-1}} - \frac{n_{f_j}}{s_j}. \tag{26}$$

Because (26) holds for any $j \in (1, l]$, it holds for $j-1$ as well, which means that $\ln(E(V_{f_{j-1}}))$ is approximated in equation (27):

$$\ln(E(V_{f_{j-1}})) \simeq -\frac{n}{m} - \frac{n_{f_1}}{s_1} \cdots - \frac{n_{f_{j-2}}}{s_{j-2}} - \frac{n_{f_{j-1}}}{s_{j-1}}. \tag{27}$$

Combining (26) and (27), the approximate value of $\ln(E(V_{f_j}))$ s given in equation (28):

$$\ln(E(V_{f_j})) \simeq \ln(E(V_{f_{j-1}})) - \frac{n_{f_j}}{s_j}. \tag{28}$$

From (28), n_{f_j} is solved in equation (29):

$$n_{f_j} \simeq s_j \ln(E(V_{f_{j-1}})) - s_j \ln(E(V_{f_j})). \tag{29}$$

By replacing $E(V_{f_j})$ and $E(V_{f_{j-1}})$ with the instance values of V_{f_j} and $V_{f_{j-1}}$ that are directly obtained from the constructed virtual arrays, B_{f_j} and $B_{f_{j-1}}$, respectively, our hierarchical virtual bitmap estimator (HVE) is given in equation (30):

$$\hat{n}_{f_j} \simeq s_j \ln(V_{f_{j-1}}) - s_j \ln(V_{f_j}), \tag{30}$$

where $\hat{n}_{f_j}$ refers to the estimate of true spread n_{f_j}. When $j = 1$, we have the spread of f_1 given in equation (31):

$$\hat{n}_{f_1} \simeq s_1 \ln(V_{f_0}) - s_1 \ln(V_{f_1}), \tag{31}$$

which is consistent with the non-hierarchical estimator of (7).

5 Performance Evaluation

In this section we evaluate the performance of the proposed Hierarchical Virtual bitmap Estimator (HVE) in comparism with prior art, through simulations on CPU and GPU implements.

We use real Internet traces from CAIDA [26] to simulate a two-level hierarchical traffic model. A second-level (base) flow consist of all packets to a destination address in downloaded traces. The spread of base flows are from the range $[1, 5000]$. Aggregate flows are identified by 16-bit prefix of the destination addresses. This classification results in about 4000 aggregate flows whose spreads are in a range $[1, 15000]$.

The most related and state-of-art compact spread estimators that we compare our work with are the virtual bitmaps (VB) [14] and opensketch(bitmap) [3] (check Section 3 for brief descriptions), which are flat single-level spread estimators. All the estimators [HVE, VB and opensketch(bitmap)] attempt to estimate a flow spread in a tight memory but their design goals are somewhat different. The main goal of the VB and opensketch(bitmap) is compact and accurate spread estimation for all types of flows, whether independent or otherwise, through sharing of memory among all flows. On the other hand, HVE tries to increase throughput (which is the number of packets processed per seconds) of hierarchical flows, while at the same time delivering compact spread estimations whose accuracy is comparable to the state-of-art. We implement VB and opsketch(bitmap) in two ways. All aggregate and base flows share the whole available physical memory in the first implementation. However, in the second implementation, one half of the physical memory is allocated to encode aggregate flows, while the other half is allocated to encode base flows. We denote the second implementation as VB* and opensketch(bitmap)*.

We will first compare the throughput of HVE with VB and opensketch(bitmap) for the setup above on CPU and GPU implementations. Afterward, we will compare the accuracy of the three estimators. Our goal is to examine whether our multi-level etimator delivers a superior throughput at the same costs of single-level estimators, that is if HVE is at least as accurate as VB and opensketch(bitmap).

Throughput of HVE: We compare the throughput of HVE with VB and opensketch(bitmap) on CPU and GPU implementations, whose specifications are the following:
CPU Implementation: We implement all the three estimators in Java, version 9, on a multi-core CPU, running Intel(R) Xeon processor @ 3.7GHz. The machine has 32GB of RAM with 2TB HDD and 512 SSD.
GPU Implementation: We use CUDA 10 toolkit to program an NVIDIA GTX 1070 GPU, with 8GB GDDR5 memory @ 1506MHz. The GPU has 1920 cuda cores.
We utilize parallel processing to speed up the online encoding on GPU implementation.

Throughput indicates the processing speed of the three estimators. We compare throughput in Table 1. Clearly, on CPU, HVE processes up to 155% more packets than VB and opensketch(bitmap). Table 1 also shows that HVE processes at least

43% more packets than VB and opensketch(bitmap) on GPU.

Table 1: The throughput (in million packets per second) of HVE, opensketch(bitmap) and VB.

Estimatord	**Software (Mpkt/sec)**	**GPU (Mpkt/sec)**
HVE	4.44	696
opensketch(bitmap)	1.74	445
opensketch(bitmap)*	1.94	423
VB	2.65	487
VB*	2.66	462

Accuracy of HVE: In the previous section, we show that HVE significantly improve throughput compared to state-of-the-art. Here, we will show that this advantage does not degrade its accuracy. In fact, we show that HVE's accuracy is up to that of VB and opensketch(bitmap) and even better for some flows.

We evaluate the accuracy of the three estimators when we allocate $m = 0.25$MB, 1MB, 2MB and 4MB. In our implementation of HVE, we set the lengths of virtual arrays for aggregate and base flows as $s_{f_1} = 64000$ bits and $s_{f_2} = 8000$ bits, respectively. These parameters are chosen to ensure that the average relative error of estimated spread of the flow in the data set above under HVE is less than 5%. However, we obtain a large average relative error when the same parameters are used under virtual bitmap and opensketch(bitmap). This because the larger a virtual bitmap is the more error it accumulates. On the other hand, we calculate the length of (virtual) bitmaps sufficient to estimate the largest spread. Consequently, the length of virtual bitmaps used to estimate the spread of aggregate and base flows under VB and VB* are 3000 bits and 1000 bits, respective. Also, the length of bitmaps used in opensketch(bitmap) and opensketch(bitmap)* is 3000 bits. In our evaluation, the spread of a large flow is at least 1000, while the spread of a small (to medium) flow is less than 1000.

We present our estimation results in Fig. 3 and 4 for aggregate and base flows, respectively, when m =2M. In Plots (a) - (e) of each figure, $x-$axis represents the actual spread of a flow and $y-$axis represents the corresponding estimated spread value. Each point on the plots represents a flow and line $y = x$ is shown for reference, such that the closer a point is to the line, the more accurate the estimation represented by the point. Plot (f) in the figures compares standard error of HVE to that of VB, VB*, opensketch(bitmap) and opensketch(bitmap)*. In the plot, the $x-$axis represents the actual flow spread while the $y-$axis represents standard error of the estimations

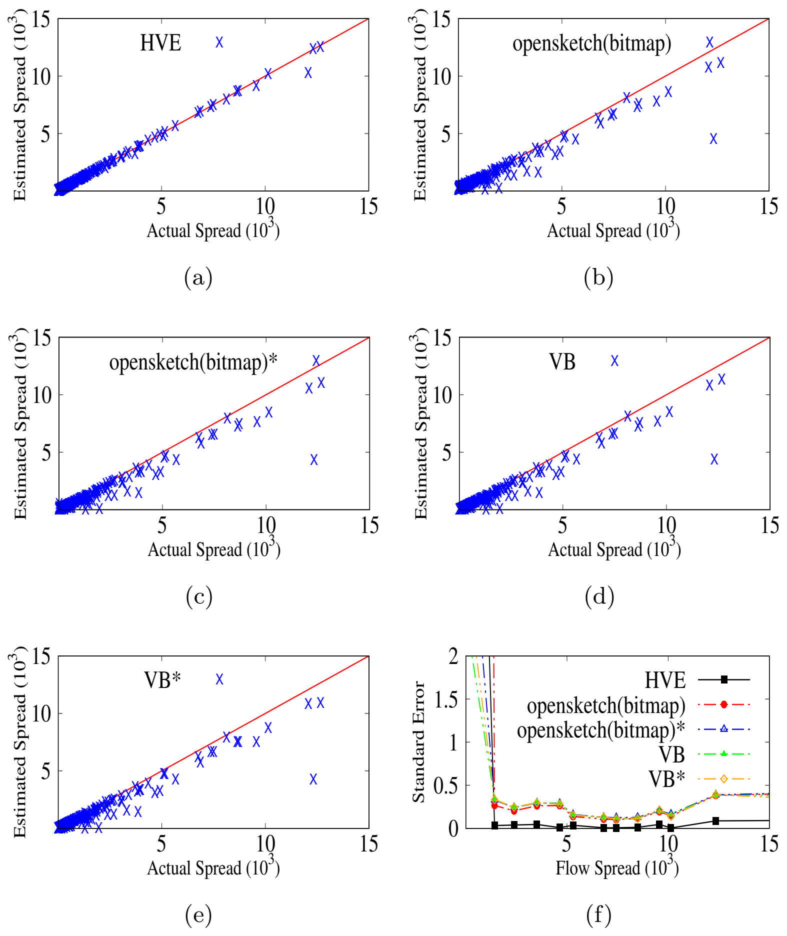

Fig. 3: Accuracy of HVE (plot (a)) vs. opensketch(bitmap) (plots (b) and (c)) vs. VB (plots (d) and (e)) for **aggregate flows**

Plot (a) in Fig. 3 and 4 shows that HVE accurately estimate the spread of aggregate and base flows, respectively, since the points cluster more closely around the equality line. For aggregate flows, Fig. 3 compares (a) HVE with (b) opensketch(bitmap), (c) opensketch(bitmap)*, (d) VB, and (e) VB* and their relative errors in Fig. 3(f). We see that HVE is significantly more accurate for large aggregate flows. We further confirm these results in Table 2, which compares the accuracy of the estimators for aggregate flows when $m = 0.25$MB, 1MB, 2MB and 4MB. The average relative error of large aggregate flows under HVE is at least 77% smaller than other estimators.

Fig. 4 compares the accuracy of base flows under (a) HVE with (b) opensketch(bitmap), (c) opensketch(bitmap)*, (d) VB, and (e) VB* and their relative er-

Table 2: Comparison of average relative error of estimated spreads of large aggregate flows (whose spreads are greater than 1000) under HVE, VB and opensketch(bitmap). Additionally, we also compare the average absolute error of estimated spreads of small aggregate flows (whose spread is less than 1000) under the estimations.

Estimators	**Relative error of large flows (spread $\geq$ 1000)**			**Absolute error of small flows (spread $<$ 1000)**		
	$m = 0.25$MB	$m = 1$MB	$m = 4$MB	$m = 0.25$MB	$m = 1$MB	$m = 4$MB
HVE	0.048	0.029	0.022	61.45	31.62	16.49
opensketch(bitmap)	0.766	0.171	0.156	1875.09	361.64	66.14
opensketch(bitmap)*	0.312	0.214	0.212	744.99	43.72	11.88
VB	0.220	0.219	0.217	25.95	17.66	13.80
VB*	0.217	0.215	0.214	22.30	16.55	13.91

rors in Fig. 4(f). Clearly, the accuracy of HVE is up to VB and opensketch(bitmap). In particular, as shown in Table 3, which compares the accuracy of the estimators for base flows when $m = 0.25$MB, 1MB, 2MB and 4MB, the average relative error of HVE for large base flows is at least 63% and up to 97% smaller than opensketch(bitmap), while its accuracy is similar to VB (in the worst case).

Although the average absolute error for small base and aggregate flows is larger for HVE compared with VB and VB*, it is still very small relative to actual spreads (about 1.7% on average, when $m = 4$MB). Note that for large aggregate and base flows, the accuracy of HVE is up to (or surpass in certain cases) the other estimators. This is important because accurate estimation of large spreads is needed for essential network management tasks, such as superspreader identification.

Table 3: Comparison of average relative error of estimated spreads of large base flows (whose spread is greater than 1000) under the HVE, VB and opensketch(bitmap). Additionally, we also compare the average absolute error of estimated spreads of small base flows (whose spread is less than 1000) the estimators.

Estimators	**Relative error of large flows (spread $\geq$ 1000)**			**Absolute error of small flows (spread $<$ 1000)**		
	$m = 0.25$MB	$m = 1$MB	$m = 4$MB	$m = 0.25$MB	$m = 1$MB	$m = 4$MB
HVE	0.021	0.018	0.013	27.41	19.77	16.59
opensketch(bitmap)	0.974	0.197	0.036	1883.84	370.48	70.031
opensketch(bitmap)*	0.431	0.142	0.081	939.26	204.91	39.74
VB	0.027	0.017	0.015	17.97	9.97	5.29
VB*	0.038	0.032	0.031	13.43	6.87	3.67

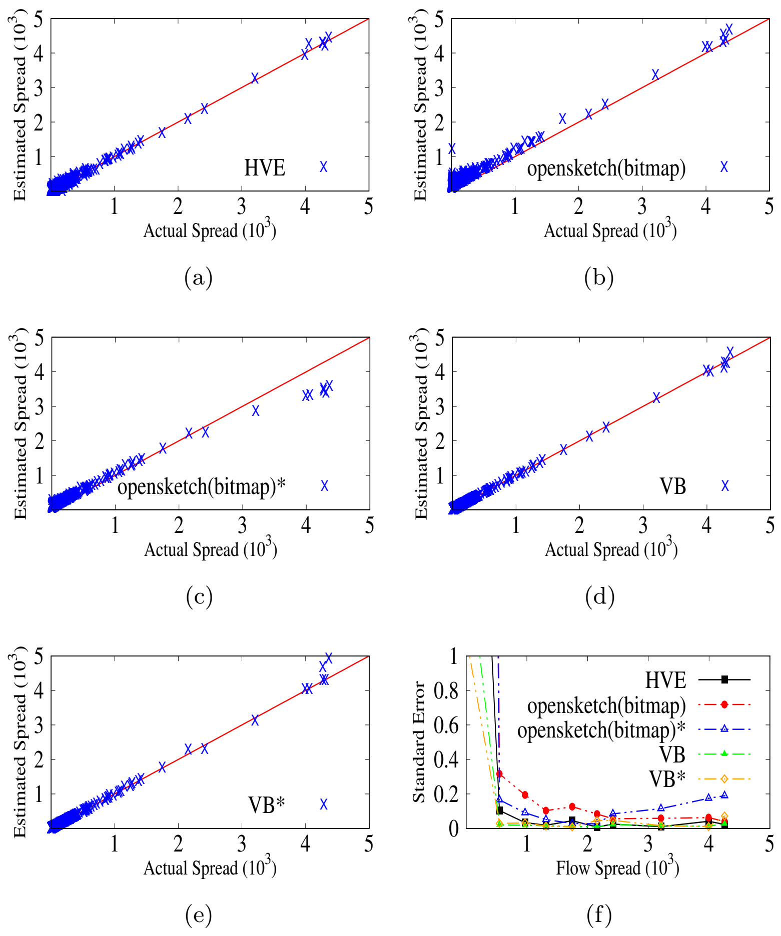

Fig. 4: Accuracy of HVE (plot (a)) vs. opensketch(bitmap) (plots (b) and (c)) vs. VB (plots (d) and (e)) for **base flows**

6 Conclusions

This paper proposes a new hierarchical measurement architecture by introducing hierarchical virtual bitmaps estimator, which extends the capability of existing single-level network traffic measurement tools and enables more efficient online data encoding of flows with hierarchical structure. We mathematically derive a hierarchical spread estimator that enables multi-level spread estimation at the same costs of single-level spread estimators. Finally, through CPU and GPU implementations, we show that our hierarchical virtual bitmap estimator's throughput significantly exceeds prior art while its accuracy is in general comparable with (or at times better than that of) prior art. The future works include rigorus analysis of HVE's accuracy, that is proving HVE unbiasness and deriving its confidence interval.

7 Acknowledgment

This work was supported by National Science Foundation of US under grants CNS-1909077 and CNS-1719222.

References

1. M. Moshref, M. Yu, and R. Govindan, "Resource/accuracy tradeoffs in software-defined measurement," in *Proceedings of HotSDN.* IEEE, 2015.
2. T. Li, S. Chen, and Y. Ling, "Fast and compact per-flow traffic measurement through randomized counter sharing," in *Proceedings of INFOCOM.* IEEE, 2011.
3. M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Symposium on Networked Systems Design and Implementation.* USENIX, 2013.
4. V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, "Csamp: A system for network-wide flow monitoring," in *Symposium on Networked Systems Design and Implementation.* USENIX, 2008.
5. Y. Yu, C. Qian, and X. Li, "Distributed and collaborative traffic monitoring in software defined networks," in *Proceedings of HotSDN.* IEEE, 2014.
6. Y. Du, H. Huang, Y. e Sun, S. Chen, and G. Gao, "Self-adaptive sampling for network traffic measurement," in *in Proceeding of IEEE INFOCOM*, 2021.
7. O. Odegbile, S. Chen, D. Melissourgos, and Y. Wang, "Accurate hierarchical traffic measurement in datacenters through differentiated memory allocation," in *Proceedings of the 6th International Conference on Big Data Computing and Communications (BigCom)*, 2020.
8. V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 15, Jul. 2009.
9. R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *Conference on Local Computer Networks.* IEEE, 2010.
10. Netflow configuration guide. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/netflow/configuration/15-mt/nf-15-mt-book.html
11. K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *in ACM Transactions on Database Systems*, vol. 15, no. 2, pp. 208–229, Jun. 1990.
12. P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," in *Proceedings of Conference on Analysis of Algorithm*, 2007.
13. D. M. Kane, J. Nelson, and D. P. Woodruff, "An optimal algorithm for the distinct elements problem," in *Symposium on Principles of database systems*, 2010, pp. 41–52.
14. M. Yoon, T. Li, S. Chen, and J. Peir, "Fit a compact spread estimator in small high-speed memory," *in IEEE/ACM Transactions on Networking*, vol. 19, no. 5, Oct. 2011.
15. Q. Xiao, S. Chen, M. Chen, and Y. Ling, "Hyper-compact virtual estimators for big network data based on register sharing," in *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems.* ACM, 2015, pp. 417–428.
16. S. Chen, Y. Zhou, and S. Chen, "Efficient hierarchical traffic measurement in software-defined datacenter networks," in *10th International Conference on Cloud Computing.* IEEE, 2017.
17. G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min," *Algorithms.*

18. Q. Xiao, S. Chen, Y. Zhou, and J. Luo, "Estimating cardinality for arbitrarily large data stream with improved memory efficiency," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 433–446, 2020.
19. H. Wang, C. Ma, O. Odegbile, S. Chen, and J.-K. Peir, "Randomized error removal for online spread estimation in data streaming," in *in Proceeding of VLDB Endowment (PVLDB)*, 2021.
20. H. Huang, Y. e Sun, C. Ma, S. Chen, Y. Zhou, W. Yang, S. Tang, H. Xu, and Q. Yan, "An efficient k-persistent spread estimator for traffic measurement in high-speed networks," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1463–1476, Aug. 2020.
21. Y. e Sun, H. Huang, C. Ma, S. Chen, Y. Du, and Q. Xiao, "Online spread estimation with non-duplicate sampling, proc. of ieee infocom," in *Proceedings of IEEE INFOCOM)*, 2020.
22. T. Yang, J. Jiang, P. Liu, and Q. Huang, "Elastic sketch: adaptive and fast network-wide measurements," in *Conference of Special Interest Group on Data Communication.* ACM, 2018, pp. 561–575.
23. Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Conference of Special Interest Group on Data Communication.* ACM, 2016, pp. 101–114.
24. Q. Xiao, Z. Tang, and S. Chen, "Universal online sketch for tracking heavy hitters and estimating moments of data streams," in *Proceedings of IEEE INFOCOM)*, 2020.
25. Y. Zhou*, Y. Zhang*, C. Ma, S. Chen, and O. Odegbile, "Generalized sketch families for network traffic measurement," in *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, 2019, (* co-first authors).
26. Caida data - overview of datasets, monitors, and reports. [Online]. Available: https://www.caida.org/data/overview/