Identification of Heavyweight Address Prefix Pairs in IP Traffic

Patrick Truong and Fabrice Guillemin
Orange Labs, 2, avenue Pierre Marzin, F-22307 Lannion, France
{patrick.truong, fabrice.guillemin}@orange-ftgroup.com

***Abstract*—We develop in this paper a new algorithm for identifying those pairs of source and destination prefixes giving rise to a significant amount of global traffic, referred to as multi-dimensional Hierarchical Heavy Hitters (mHHH). We represent the source and destination address pairs by weighted circuits in a graph so that the prefix pairs are considered as groups of circuits in the graph. Identifying mHHH pairs then consists in finding clusters of circuits that have a cumulative weight greater than an user specified threshold. Starting from this model, we propose an off-line algorithm to exhaustively find mHHH pairs. On the basis of this exhaustive algorithm, we introduce an efficient on-line algorithm that identifies mHHH pairs in real time with provable accuracy and memory guarantees. Experimental results with real traffic data from France Telecom networks illustrate the efficiency of the algorithm.**

***Index Terms*—Heavy Hitters, Security, Patricia Trees**

I. Introduction

With the rapid increase of link transmission capacities and the proliferation of new applications, the Internet is becoming the integrated multiservice communication network supporting all kinds of services. This situation raises many issues with regard to quality of service issues and security. The Internet suffers from many vulnerabilities, which can be exploited by malicious users to disturb services supported by machines connected to the Internet.

In the framework of security, one basic task consists in detecting heavy hitters, that is, data flows with a volume larger than a given proportion of global traffic, a flow being usually identified by means of information elements in the packet header (e.g., IP source and destination addresses, port numbers, and protocol). Beyond heavy hitter flows, it is also necessary to aggregate several flows according to the natural hierarchical structure induced by the IP addressing scheme, giving rise to Hierarchical Heavy Hitters (HHHs); see [1], [2], [3] for instance. This type of heavy hitters naturally arises for instance in the detection of SYN flood attacks.

HHH identification consists in aggregating individual IP addresses into prefixes. HHHs are recursively defined as prefixes that have a volume greater than a portion of total network traffic, even after removing the volumes of the descendants that are already identified as HHHs; these quantities are referred to as *discounted volumes* by Cormode *et al* [4]. Several algorithms have been proposed in the technical literature to detect HHHs; see for instance [4], [1], [2], [3].

However, attacks are often distributed and involve a large number of sources and destinations. This is typically the case of distributed DoS attacks, which can be detected only by examining pairs of IP source and destination addresses. This leads to the definition of Multidimensional Hierarchical Heavy Hitters (mHHHs) that are source and destination IP prefix pairs concentrating a volume greater than an operator specified threshold. The concept of mHHH has been introduced by Estan *et al.* [5] and Cormode *et al* [4], [6]. In [5], the authors present some heuristics for off-line (m)HHH computation.

The concept of mHHH is a generalization of that of HHH over multiple hierarchies. In the context of IP data, mHHHs are source and destination prefix pairs that have a discounted volume greater than a fraction of global traffic. As a prefix pair can have two parents, the volume of an address pair can be aggregated into several distinct mHHHs. This phenomenon can generate overcounting errors when computing mHHHs as well explained Cormode *et al* [6]. Moreover, given the huge dimension of the addressing space ($2^{32} \times 2^{32}$), finding mHHHs can rapidly lead to a combinatorial explosion of the number of states to analyze (see [7] for a study of the space complexity). The mHHH problem is therefore more difficult than the HHH one.

Cormode *et al.* [6] have proposed an algorithm for on-line mHHH identification. Their algorithm generalizes the Lossy Counting algorithm [8] in order to take into account the hierarchical levels by using a lattice-like data structure to represent prefix pairs, but it suffers from rather high false positive ratios due to overcounting errors and fails bounding the estimation of the discounted volumes. This is why we introduce in this paper an new algorithm reducing both positive and negative false ratios.

The organization of this paper is as follows. We first introduce in Section II a new formulation of the mHHH problem using some basic notions in graph theory. This novel approach models prefix pairs as groups of circuits in a graph and allows us to cancel the overcounting problem when computing the discounted volumes. In Section III, we present an off-line algorithm that exhaustively finds mHHHs with their exact volume. For data stream applications, we propose in Section III an on-line algorithm that finds approximate mHHHs with guarantees for accuracy and space efficiency. Section V provides experimental validations of our on-line algorithm, and we conclude this paper in Section VI.

II. Problem Formulation

A. Definitions

Let us denote by A_s and A_d the sets of source and destination IP addresses. Theses sets inherit the IP address hierarchy and each of them can be composed of up to 2^{32} elements. We can model the set A_s (resp. A_d) by a binary tree $\mathcal{T}_s$ (resp. $\mathcal{T}_d$). The nodes of the trees are IP prefixes and the leaves correspond to individual hosts. The root of the tree $\mathcal{T}_s$ (resp. $\mathcal{T}_d$), denoted by o_s (resp. o_d), represents all the source (resp. destination) prefixes. The trees $\mathcal{T}_s$ and $\mathcal{T}_d$ are equipped with the partial order defined by: $p \preceq q$ if and only if q is a prefix of p (i.e. p is a descendant of q) or $p = q$.

The input stream of the mHHH problem is a sequence $\mathcal{F} = \left\langle (e_1^{(1)}, e_2^{(1)}), (e_1^{(2)}, e_2^{(2)}), \dots \right\rangle$ of source and destination IP address pairs, i.e. elements from the set $\Omega = A_s \times A_d$. The frequency or volume f_e of an element e in $\mathcal{F}$ is the number of counts of the element e in $\mathcal{F}$. The size or total volume of the stream $\mathcal{F}$, i.e., the total number of elements in $\mathcal{F}$, is $N = \sum_{e \in \Omega} f_e$.

By using the hierarchy on the prefixes, we define a hierarchy on the domain Ω by using the partial order defined by: $(p_1^{(1)}, p_2^{(1)}) \preceq (p_1^{(2)}, p_2^{(2)})$ if and only if $p_1^{(1)} \preceq p_1^{(2)} \wedge p_2^{(1)} \preceq p_2^{(2)}$. This order allows us to represent the elements of Ω by a lattice. Each node p in the lattice is a source and destination prefix pair (p_1, p_2).

We say that the node $p = (p_1, p_2)$ belongs to the level l of the hierarchy if p_1 (resp. p_2) belongs to the level l_1 (resp. l_2) in the tree $\mathcal{T}_s$ (resp. $\mathcal{T}_d$) such as $l = l_1 + l_2$ (i.e. the sum of the lengths of the two prefixes is l). There are altogether 65 distinct levels in the hierarchy. The first level 0 is the root of the lattice, denoting all the prefix pairs, and the elements of Ω are the leaves of the lattice at the last level $L = 64$. The sublattice issued from the node p, denoted by $S(p)$, aggregates all the leaves of the lattice descendant from p. The *cumulative volume* of p is then the sum of the volumes of all its descendant leaves.

Let us recall the mHHH definition given by Estan *et al.* [5] and Cormode *et al* [6].

Definition 1: Let $\phi \in [0, 1]$, the ϕ-multidimensional Hierarchical Heavy Hitters (mHHHs) are defined recursively as follows:

- The set $\mathcal{H}_L$ contains the heavy hitters of the stream $\mathcal{F}$, i.e., those IP address pairs e such that $f_e \geq \lfloor \phi N \rfloor$.
- For $\ell < L$, the *discounted volume* of a node (or equivalently a pair of prefixes) p at level ℓ is defined by

$$f_p = \sum_{e \in \mathcal{F} \cap (S(p) \setminus \mathcal{H}_{\ell+1})} f_e,$$

  where the sets $\mathcal{H}_\ell$ are defined by

$$\mathcal{H}_\ell = \mathcal{H}_{\ell+1} \cup \{p \text{ at level } \ell : f_p \geq \lfloor \phi N \rfloor\}$$

- The ϕ-mHHHs of the stream $\mathcal{F}$ are the elements of the set $\mathcal{H}_0$.

While the cumulative volume of a prefix pair p aggregates all the volumes of the address pairs descendant from p, the discounted volume of p aggregates only the volumes of those descendant address pairs not included in any mHHH that is a strict descendant of p. The mHHHs are then those prefixes whose discounted volume is greater than $\lfloor \phi N \rfloor$.

A prefix pair can have up to two parents so that the discounted volumes of distinct mHHHs can overlap in the sense that the volume of a same leaf can be counted into these distinct mHHHs. The basic problem of the on-line algorithms in the literature is that the volumes of the various nodes have to be rolled up to those of their parents in the lattice. We easily see that this operation can rapidly lead to overcounting errors when computing the discounted volumes (see [6] for more details).

B. Circuits

To limit overcounting errors, we formalize a novel approach to the mHHH problem by using graph theory. We model the lattice by a weighted multidigraph $G = (V, E)$. The vertices are the nodes of the trees $\mathcal{T}_s$ and $\mathcal{T}_d$, plus a specific element o_r so that there exist two arcs from o_r to the root o_s of $\mathcal{T}_s$ and from the root o_d of $\mathcal{T}_d$ to o_r. The vertex o_r is only used to connect the two trees. Two vertices x and y are connected by an arc from x to y with weight v if there exist an address pair (a_1, a_2) in the stream $\mathcal{F}$ having a volume $f_{(a_1,a_2)} = v$ and satisfying one of the following conditions:

- $(x = o_r)$ and $(y = o_s)$;
- $(x = o_d)$ and $(y = o_r)$;
- $(x = a_1)$ and $(y = a_2)$;
- x and y are prefixes of a_1, and y is a direct descendant of x, i.e. if x belongs to level l in the tree $\mathcal{T}_s$, y is at level $l + 1$ of $\mathcal{T}_s$;
- x and y are prefixes of a_2, and x is a direct descendant of y in $\mathcal{T}_d$.

We observe that there are as many circuits in the graph G as leaves in the lattice. We can actually associate each IP address pair $(p_1^{(32)}, p_2^{(32)})$ of volume $f_{(p_1,p_2)}$ with a circuit of weight $f_{(p_1,p_2)}$ and traversing the origin vertex o_r; we denote this circuit by $C_{(p_1,p_2)} = (o_r, o_s, p_1^{(1)}, \dots, p_1^{(31)}, p_1^{(32)}, p_2^{(32)}, p_2^{(31)}, \dots, p_2^{(1)}, o_d, o_r)$. Let $\mathcal{C} = \{C_1, \dots, C_m\}$ be the set of circuits in the graph. We identify each circuit by the leaf (= pair of address) it represents in the lattice: $C_i = (p_{1,i}^{(32)}, p_{2,i}^{(32)})$. The sum of the weights of the circuits is the total volume N of the stream $\mathcal{F}$.

We define an operation $\vee$ on the set $\mathcal{C}$ of circuits in the graph. Consider two circuits $C_i = (p_{1,i}^{(32)}, p_{2,i}^{(32)})$ and $C_j = (p_{1,j}^{(32)}, p_{2,j}^{(32)})$ of weights f_i and f_j. We denote by $p_1^{(l)}$ (resp. $p_2^{(k)}$) the longest prefix match between the two addresses $p_{1,i}^{(32)}$ and $p_{1,j}^{(32)}$ (resp. $p_{2,i}^{(32)}$ and $p_{2,j}^{(32)}$). We define the join of the circuits C_i and C_j as a circuit $C_i \vee C_j$ of weight $f_i + f_j$:

$$C_i \vee C_j = (o_r, o_s, p_1^{(1)}, \dots, p_1^{(l)}, p_2^{(k)}, \dots, p_2^{(1)}, o_d, o_r)$$

Thus, the join of circuits is the commutative law $\vee$ of the lattice in its algebraic structure definition, so that each node in the lattice can be viewed as a join of circuits in the graph.

We say that the join R is included in the join $R^{'}$, and we note $R \subseteq R^{'}$, if all the circuits composing R are joined in R'. Finally, the join of R and $R^{'}$ is the join of circuits composing R and $R^{'}$.

As a circuit in the graph is also a join of itself, we define the level of a join R of circuits as $|R|-3$, where $|R|$ is the length of the circuit denoting the join R. This definition mades the levels of joins correspond to the hierarchy levels in the lattice.

Let φ: $\mathcal{T}_s \times \mathcal{T}_d \rightarrow \mathcal{P}(\mathcal{C})$ be the function defined by $\varphi(v_s, v_d) = \{C_i | v_s \in C_i \text{ and } v_d \in C_i\}$, i.e. for any pair of nodes in the trees $\mathcal{T}_s$ and $\mathcal{T}_d$, we associate the set of circuits traversing these nodes. The *discounted volume* of a prefix pair (p_1, p_2) is defined by

$$f_{(p_1,p_2)} = \sum_{C_i \in \varphi(p_1,p_2) \setminus \{C_j | \exists (u,v) \in \mathcal{H}_{\ell+1},\ (u,v) \preceq (p_1,p_2),\ u \in C_j \wedge v \in C_j\}} f_{C_i},$$

where the sets $\mathcal{H}_\ell$ are defined by

$$\mathcal{H}_\ell = \mathcal{H}_{\ell+1} \cup \left\{(p_1, p_2) \text{ at level } \ell : f_{(p_1,p_2)} \geq \lfloor \phi N \rfloor\right\}$$

The mHHH definition can then be reformulated by means of circuits in the graph G as follows:

Definition 2: [using circuits in the graph] Let $\phi \in [0, 1]$, the ϕ-multidimensional Hierarchical Heavy Hitters (mHHHs) are defined recursively as follows:

- At level $L = 64$, the set $\mathcal{H}_L$ contains the heavy hitters of the stream $\mathcal{F}$, i.e., the circuits C_i that have a weight f_{C_i} greater than $\lfloor \phi N \rfloor$.
- At level $\ell < L$, the mHHHs are the joins of circuits that have a weight greater than $\lfloor \phi N \rfloor$ even after removing all the descendant joins that are already mHHHs. We define the set

$$\begin{aligned}\mathcal{H}_\ell = &\mathcal{H}_{\ell+1} \cup \left\{R = \vee C_i \text{ at level } \ell | \right. \\ &\left. (f_R \geq \lfloor \phi N \rfloor) \wedge \left(\nexists R^{'} \in \mathcal{H}_{\ell+1}, R^{'} \subseteq R\right)\right\}\end{aligned}$$

- The ϕ-mHHHs of the stream $\mathcal{F}$ are the elements of the set $\mathcal{H}_0$. The weights of the joins in $\mathcal{H}_0$ are exactly the discounted volumes of the mHHHs.

Definition 2 allows us to model any prefix pair as a join of circuits in the graph so that the weight of the join is exactly the discounted volume of the prefix pair. We will use this model in the rest of the paper. Modeling the mHHH problem with circuits in a graph enables us to cancel the overcounting problem as shown in the next sections.

III. Off-line Detection of mHHHs

We present in this section an off-line algorithm that exhaustively finds mHHHs with their exact discounted volumes.

From Definition 2, we can make the following statement. Let R be an mHHH join at level l representing the prefix pair (p_1, p_2) and let us denote by j (resp. k) the length of p_1 (resp. p_2). We have $l = j+k$. If a circuit C_i is in R, then it cannot be included in any other join at higher level $l^{'} = j^{'} + k^{'}$ such as $l^{'} < l$ and $j^{'} \leq j \wedge k^{'} \leq k$. This simple observation makes it possible to compute the mHHH joins by describing the trees $\mathcal{T}_s$ and $\mathcal{T}_d$ in postorder. We actually describe the tree $\mathcal{T}_s$ in postorder, and for each visited node, we describe the tree $\mathcal{T}_d$ in postorder. Figure 1 illustrates this procedure.

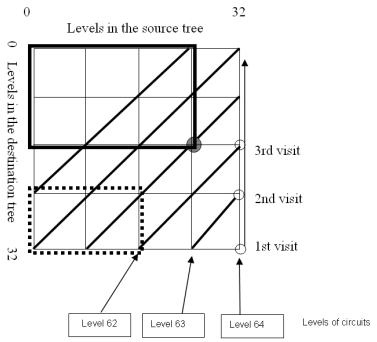


Fig. 1. Illustration of the inspection of the trees $\mathcal{T}_s$ and $\mathcal{T}_d$.

When describing the trees, if a prefix pair at level (j, k) is an mHHH, the circuits, that compose the join denoting the prefix pair, cannot be grouped into any join located at a superior level $(j^{'}, k^{'})$ such as $j^{'} \leq j$ and $k^{'} \leq k$ (black rectangle on Figure 1), but they can possibly be aggregated in the joins at level $(j^{'}, k^{'})$ such as $j^{'} < j$ and $k^{'} > k$ (dotted black rectangle on Figure 1). To overcome this problem, we define the reduced form of a circuit.

We define the k-th reduced form of a circuit $C_i = (o_r, o_s, p_1^{(1)}, \ldots, p_1^{(31)}, p_1^{(32)}, p_2^{(32)}, p_2^{(31)}, \ldots, p_2^{(1)}, o_d, o_r)$, denoting an address pair $(p_1^{(32)}, p_2^{(32)})$, as a circuit $C_i^{(k)} = (o_r, o_s, p_1^{(1)}, \ldots, p_1^{(31)}, p_1^{(32)}, p_2^{(32)}, p_2^{(31)}, \ldots, p_2^{(k)}, o_r)$, for $1 \leq k \leq 32$. The k-th reduced form of the circuit C_i prevents from the aggregation of that circuit in any join at level $(j^{'}, k^{'})$ with any $j^{'}$ and $k^{'} < k$, while allowing it to be aggregated in a join at level $(j^{'}, k^{'})$ with any $j^{'}$ and $k^{'} \geq k$.

The algorithm for computing mHHHs off-line consists in describing the tree $\mathcal{T}_s$ recursively in postorder, starting from the root node:

1) Describe the left subtree.
2) Describe the right subtree.
3) Inspect the node p_1 at level j: describe the tree $\mathcal{T}_d$ in postorder:
   - 3.1) Describe the left subtree.
   - 3.2) Describe the right subtree.
   - 3.3) Inspect the node p_2 at level k. Take the join R of all the circuits traversing p_1 and p_2. The discounted volume of (p_1, p_2) is the weight of R.
     - i) If $f_R \geq \lfloor \phi N \rfloor$, (p_1, p_2) is an mHHH.
       - If we are inspecting a leaf node of $\mathcal{T}_d$, i.e. the length of p_2 is $k = 32$, we remove from the graph all the circuits composing the join R.
       - If we are inspecting an internal node, i.e. $k < 32$, we replace all the circuits in the join R with their $(k+1)$-th reduced form.
     - ii) Otherwise,(p_1, p_2) is not an mHHH, we do nothing and go to the next step of the algorithm.

For more thoroughly explaining the algorithm, let use examine the case 3.3.i when R is an mHHH. We need to remove the circuits in R from the upper levels :

- If $k = 32$, the circuits composing the join R cannot be aggregated at levels $(j^{'}, k^{'})$ with $j^{'} \leq j$ and $k^{'} \leq k$. Owing to the specific way of describing the trees, these levels correspond to the next steps of the algorithm, so all the circuits of R can be removed.
- When $k < 32$, in addition to remove the circuits in R from the joins at levels $(j^{'}, k^{'})$ with $j^{'} \leq j$ and $k^{'} \leq k$, we also need to allow them to be regrouped later in the joins at level $(j^{'}, k^{'})$ such as $j^{'} < j$ and $k^{'} > k$. That is why we use the $(k+1)$-th reduced form of the circuits.

To illustrate the algorithm, let us consider the situation represented by Figure 2. We have numbered the nodes of the source and destination trees, so that each prefix is denoted by its number. The circuits, representing address pairs, are identified by their weight, that are also the volumes of the address pairs. The mHHH threshold is set to 10.

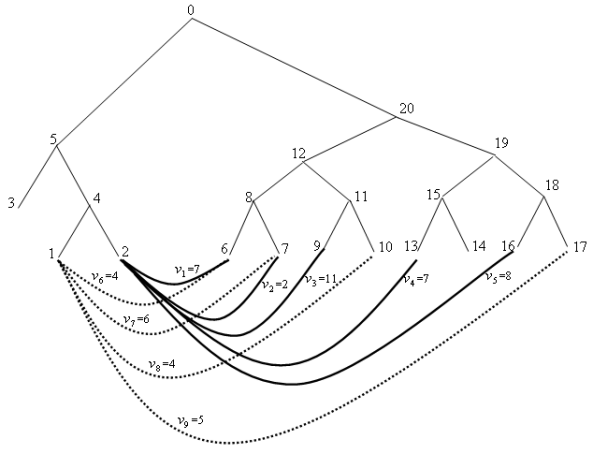


Fig. 2. Example of a graph of circuits: the nodes are named by numbers and the circuits are represented by their weight v_i

In order to keep the paper not excessively long, we limit ourselves to give the list of circuits in the graph before and after inspecting each node of the source tree, in the order of inspection (the notation $\widetilde{C_i}$ represents a reduced form of a circuit).

Node 1 The circuits traversing Node 1 before inspection are:

- $C_6 = v_6 = (0, 5, 4, 1, 6, 8, 12, 20, 0)$,
- $C_7 = v_7 = (0, 5, 4, 1, 7, 8, 12, 20, 0)$,
- $C_8 = v_8 = (0, 5, 4, 1, 10, 11, 12, 20, 0)$,
- $C_9 = v_9 = (0, 5, 4, 1, 17, 18, 19, 20, 0)$.

The only mHHH found at this node is the prefix pair (1,8) with discounted volume $v_6 + v_7 = 10$.

The circuits traversing Node 1 after inspection are:

- $\widetilde{C_6} = v_6 = (0, 5, 4, 1, 6, 0)$,
- $\widetilde{C_7} = v_7 = (0, 5, 4, 1, 7, 0)$,
- $C_8 = v_8 = (0, 5, 4, 1, 10, 11, 12, 20, 0)$,
- $C_9 = v_9 = (0, 5, 4, 1, 17, 18, 19, 20, 0)$.

Node 2 For Node 2, the prefix pairs (2,9) and (2,19) are mHHHs with respective discounted volumes $v_3 = 11$ and $v_4 + v_5 = 15$. We apply the same procedure as above and after inspection, the circuits traversing this node are:

- $C_1 = v_1 = (0, 5, 4, 2, 6, 8, 12, 20, 0)$,
- $C_2 = v_2 = (0, 5, 4, 2, 7, 8, 12, 20, 0)$,
- $\widetilde{C_4} = v_4 = (0, 5, 4, 2, 13, 15, 0)$,
- $\widetilde{C_5} = v_5 = (0, 5, 4, 2, 16, 18, 0)$.

We note that at this step, we have removed the circuit C_3 from the graph.

Node 4 We give more details for this node since the situation is more involved. The circuits before inspection are:

- $\widetilde{C_6} = v_6 = (0, 5, 4, 1, 6, 0)$,
- $\widetilde{C_7} = v_7 = (0, 5, 4, 1, 7, 0)$,
- $C_8 = v_8 = (0, 5, 4, 1, 10, 11, 12, 20, 0)$,
- $C_9 = v_9 = (0, 5, 4, 1, 17, 18, 19, 20, 0)$,
- $C_1 = v_1 = (0, 5, 4, 2, 6, 8, 12, 20, 0)$,
- $C_2 = v_2 = (0, 5, 4, 2, 7, 8, 12, 20, 0)$,
- $\widetilde{C_4} = v_4 = (0, 5, 4, 2, 13, 15, 0)$,
- $\widetilde{C_5} = v_5 = (0, 5, 4, 2, 16, 18, 0)$.

We describe now the destination tree in postorder. We first inspect Node 6. The circuits traversing Nodes 4 and 6 are $C_1 = v_1 = (0, 5, 4, 2, 6, 8, 12, 20, 0)$ and $\widetilde{C_6} = v_6 = (0, 5, 4, 1, 6, 0)$. The join of C_1 and $\widetilde{C_6}$ has a weight $v_1 + v_6 = 11$, greater than the mHHH threshold. The prefix pair $(4, 6)$ is then an mHHH. Since Node 6 is a leaf, we can completely remove the circuits C_1 and $\widetilde{C_6}$ from the graph.

Let us move to the next node (Node 7). The circuits traversing Nodes 4 and 7 are $\widetilde{C_7} = v_7 = (0, 5, 4, 1, 7, 0)$ and $C_2 = v_2 = (0, 5, 4, 2, 7, 8, 12, 20, 0)$. The join of these two circuits has a weight $v_7 + v_2 = 8$, so it is not an mHHH.

By progressing in the destination tree bottom-up, we arrive at Node 20. There exist three circuits traversing Nodes 4 and 20: C_2, C_8 and C_9. Their join gives a weight $v_2 + v_8 + v_9 = 11$, so the prefix pair $(4, 20)$ is an mHHH. We then replace these circuits by their reduced form.

Finally, the circuits traversing the node 4 after inspection are:

- $\widetilde{C_7} = v_7 = (0, 5, 4, 1, 7, 0)$,
- $\widetilde{C_8} = v_8 = (0, 5, 4, 1, 10, 11, 12, 0)$,
- $\widetilde{C_9} = v_9 = (0, 5, 4, 1, 17, 18, 19, 0)$,
- $\widetilde{C_2} = v_2 = (0, 5, 4, 2, 7, 8, 12, 0)$,
- $\widetilde{C_4} = v_4 = (0, 5, 4, 2, 13, 15, 0)$,
- $\widetilde{C_5} = v_5 = (0, 5, 4, 2, 16, 18, 0)$.

IV. On-Line Algorithm for mHHH Identification

On the basis of the exhaustive algorithm, we present in this section a computational and memory-efficient algorithm for on-line analysis.

A. Tree Data Structure

As in [2], we maintain two Patricia-like trees T_S and T_D to represent source and destination prefixes and whose update is similar to the one in a Patricia tree but is controlled by an expansion smoother T_{smooth} to limit the creation of nodes in the trees. The source tree T_S is used to maintain source prefixes. Each node has the following fields, exactly as in [2]:

- `uPrefix`: the source IP prefix represented by the node,
- `uPrefixLen`: the IP prefix length,
- `tVolume`: the estimated cumulative volume of the source prefix,
- `pLeft` and `pRight` : pointers towards the children.

The structure of the destination tree T_D is slightly different: each node y of T_D has an additional pointer towards a Patricia-like tree $\widehat{T}_y$ that we refer to as the *local source tree* associated with the node y. The local tree $\widehat{T}_y$ maintains the source prefixes that are paired with the destination prefix represented by the node y. We then have the following fields for each node of the destination tree T_D:

- `uPrefix`: the destination IP prefix represented by the node,
- `uPrefixLen`: the IP prefix length,
- `tVolume`: the estimated cumulative volume of the destination prefix,
- `pLeft` and `pRight` : pointers towards the children,
- `pTrie`: pointer towards the local source tree that maintains the source prefixes paired with the destination prefix `uPrefix`.

The node structure of a local source tree $\widehat{T}_y$ is the same as for the global source tree T_S.

The on-line algorithm uses two user-supplied parameters:

- the threshold ϕ ($0 < \phi < 1$) for mHHH detection and the error for discounted volume estimation,
- ε, such as $0 < \varepsilon < 1$ and $\varepsilon < \phi$,
- the *expansion smoother*, defined by $T_{smooth} = \frac{\varepsilon N}{L}$, with N as the total traffic volume during the observation period and $L = 64$ as the maximum level in the circuit hierarchy (see [2] for details).

B. Update Operation

Upon arrival of a packet, the global source tree T_S and the local source trees $\widehat{T}_y$ associated with nodes in the destination tree are updated with the source address of the packet as described in [2]. The update procedure for the destination tree T_D with the destination address of the packet slightly differs as we have to create relations between T_D and local source trees. Therefore, we only describe here how to update the destination tree T_D and how to link the local source trees with nodes in T_D. For this purpose, we use a C-like language notation to refer to the fields of a node y in T_D:

- `y->uPrefix` is the destination IP prefix represented by the node p.
- `y->uPrefixLen` is the length of the prefix.
- `y->tVolume` is the estimated cumulative volume of the prefix.

Let us consider a packet arrival with source address `SrcAdd`, destination address `DstAdd` and packet size v. If the tree T_D is nil (which means that we are processing the first packet), we create the first node `y` in T_D and we create a local source tree $\widehat{T}_y$ with only one node z. We initialize the different fields of y and z as follows:

- `z->uPrefix = SrcAdd`
- `z->uPrefixLen = 32`
- `z->tVolume` = v
- `y->uPrefix = DstAdd`
- `y->uPrefixLen = 32`
- `y->tVolume` = v
- `y->pTrie = z`

If T_D is not nil, we walk down the tree until we reach the *most neighbouring node* to the destination address `DstAdd`, defined as the node in the tree whose prefix contains the Longest Prefix Match (LPM) with `DstAdd`: we note `y` this node, `LPMPrefix` the longest prefix match between `y->uPrefix` and `DstAdd`, and `LPMPrefixLen` the length of the longest prefix match. There are two possible cases:

1) If `LPMPrefixLen < y->uPrefixLen` (meaning that in the case of a traditional Patrica tree, we should create the LPM node) then:
   a) If $(\texttt{y->tVolume} + v) \leq T_{smooth}$, we update the local source tree, $\widehat{T}_y$ = `y->pTrie`, of `y` with the source address `SrcAdd` and the volume v in the same way as described in [2], and we change the values of the other fields of `y`:
      - `y->uPrefix = LPMPrefix`,
      - `y->uPrefixLen = LPMPrefixLen`,
      - `y->tVolume = y->tVolume +` v,
   b) Otherwise we create the LPM node, denoted by `x` = <`LPMPrefix`, `LPMPrefixLen`>, and we initialize its volume counter as follows: `x->tVolume = y->tVolume +` v. We then distinguish three sub-cases:
      i) If $v \leq T_{smooth}$, the local source tree $\widehat{T}_x$ of the LPM node `x` is initialized with a copy of the local source tree of the node `y`, and then we update $\widehat{T}_x$ with the source address `SrcAdd`. We finally attach the child node `y` to the LPM node `x` according to the value of the (`LPMPrefixLen`+1)th bit of `y->uPrefix`.
      ii) Else if `y->tVolume` $\leq T_{smooth}$, we update the local source tree $\widehat{T}_y$ of `y` with the address `SrcAdd` and we assign this tree to the LPM node `x` (`x->pTrie = y->pTrie`). We next change the values of the following fields of `y`:
         - `y->uPrefix = DstAdd`,
         - `y->uPrefixLen = 32`,
         - `y->tVolume` = v,

         and we create a new source tree for `y` with only one node representing the address `SrcAdd`. We finally attach the node `y` as a child of the LPM node `x`
      iii) Else we create a new node, `y'`:
         - `y'->uPrefix = DstAdd`,
         - `y'->uPrefixLen = 32`,
         - `y'->tVolume` = v,

         and we create a source tree for `y'` with only one node representing the address `SrcAdd`. We

assign a copy of the local source tree of y to the node x, and we next update $\widehat{T}_x$ with the address SrcAdd. We finally attach the two nodes y and y' as the children of the LPM node x.

2) If LPMPrefixLen = y->uPrefixLen, then:
   a) If LPMPrefixLen = 32, this simply means that the address DstAdd is already present in the tree T_D and is identified by the node y: we increment its volume counter (y->tVolume += v) and we update the tree $\widehat{T}_y$ with the address SrcAdd.
   b) Else if (y->tVolume + v) $\leq T_{smooth}$, we just need to increment the volume counter of y: y->tVolume += v and update the tree $\widehat{T}_y$ with the address SrcAdd.
   c) Otherwise we create a new node, y':
      - y'->uPrefix = DstAdd,
      - y'->uPrefixLen = 32,
      - y'->tVolume = v.

      We create a source tree for y' with only one node representing the address SrcAdd, and we attach y' as a child of y.

Note that while walking down the tree T_D to find the most neighboring node, we also update the volume counter and the source tree of each visited internal node.

The counter tVolume of each node p_1 in the local source tree associated with a node p_2 in the destination tree T_D gives an estimation of the cumulative volume of the prefix pair (p_1, p_2), i.e. the cumulative weight of all the circuits traversing the nodes p_1 and p_2. The use of the expansion smoother $T_{smooth} = \frac{\varepsilon N}{L}$ ensures that this cumulative weight estimation is less than the true value by at most εN.

Let us denote $H = 32$ the height of the IP addressing hierarchy. For any $l_1 \leq H$ and $l_2 \leq H$, we cannot have more than N/T_{smooth} nodes at level l_1 in local source trees associated with nodes at level l_2 in the destination tree, otherwise the sum of the volumes of all these nodes would be greater than N. Therefore there is $O(H^2 \cdot N/T_{smooth}) = O(2H^3/\varepsilon)$ nodes in the tree structure T_D. The source tree T_S contains $O(2H^2/\varepsilon)$. Finally the data structure of our algorithm uses approximately $O(2H^3/\varepsilon)$ nodes.

C. mHHH Search

The counter tVolume of each node p_1 (resp. p_2) in the source (resp. destination) tree T_S (resp. T_D) stores the cumulative weight of all the circuits traversing p_1 (resp. p_2). We have seen in Section II that a prefix pair is modeled by a join of circuits so that the weight of the join is exactly the discounted volume of the prefix pair. Therefore, if the prefix pair (p_1, p_2) is an mHHH, meaning that its discounted volume is greater than the mHHH threshold $\lfloor \phi N \rfloor$, the counters tVolume of p_1 and p_2 have values necessarily greater than $\lfloor \phi N \rfloor$. As a consequence, when finding mHHHs, we can restrict ourselves to those nodes in the source and destination trees T_S and T_D that have a counter tVolume greater than $\lfloor \phi N \rfloor$. At the end of the observation period, we then clean up each tree by starting from the root to the leaves and deleting all the nodes that have a volume less than the mHHH threshold. This allows us to remove the superfluous nodes and only keep those which can potentially give rise to mHHHs.

We then perform the mHHH search on the cleaned trees T_S and T_D. The operation is somewhat similar to the one in the off-line algorithm of Section III: we describe the cleaned source tree T_S in postorder from the root node, and for each inspected node, we describe the cleaned destination tree T_D in postorder. When inspecting a node p_1 at level j in tree T_S, we describe the cleaned tree T_D in postorder. When inspecting Node p_2 at level k in T_D, we look up the node that is an immediate descendant of the prefix p_1 in the local source tree $\widehat{T}_{p_2}$ associated the node p_2. If such an node does not exist, the discounted volume of (p_1, p_2) is nil (there is no circuit traversing p_1 and p_2), and we go to the next step of the algorithm. Otherwise, let $\widehat{p}_1$ be this node. The estimation of the discounted volume of (p_1, p_2) is then given by $\widehat{p}_1$->tVolume.

   i) If $\widehat{p}_1$->tVolume $\geq \lfloor \phi N \rfloor$, the prefix pair (p_1, p_2) is an mHHH and we need to remove the circuits composing (p_1, p_2) from upper levels (pseudocode in Table I).
   ii) Otherwise,(p_1, p_2) is not an mHHH, we do nothing and go to the next step of the algorithm.

Let us detail the case when (p_1, p_2) is an mHHH. We denoted by $sub\widehat{T}_{p_2}$ the subtree of $\widehat{T}_{p_2}$ whose root is $\widehat{p}_1$. The leaves of $sub\widehat{T}_{p_2}$ are actually the circuits composing the mHHH join (p_1, p_2), and the weight of this join (i.e. the discounted volume of (p_1, p_2)) is estimated by the volume of $sub\widehat{T}_{p_2}$, i.e. $V = \widehat{p}_1$->tVolume. Due to the compression smoother when updating the tree data structures, using the sum of the volumes of all the leaves of $sub\widehat{T}_{p_2}$ as an estimation of the discounted volume is less accurate than V (the sum is smaller than V because of missed volumes), so that in addition to remove the leaves from the local source trees associated with all the parent nodes of p_2 in T_D (including the local tree $\widehat{T}_{p_2}$ of p_2), we also need to remove the difference between V and the cumulative volume of the leaves of $sub\widehat{T}_{p_2}$.

We start by removing the subtree $sub\widehat{T}_{p_2}$ from the tree $\widehat{T}_{p_2}$ and as a consequence, we also discount the removed volume $\widehat{p}_1$->tVolume in the volume counters of all the parent nodes of $\widehat{p}_1$.

Table I gives the code for the removal of $sub\widehat{T}_{p_2}$ from the local source trees of the parent nodes of p_2 in T_D. Let $\mathcal{C}$ be the list of the leaf nodes of $sub\widehat{T}_{p_2}$. We inspect the parent nodes of p_2 in postorder and perform the following operations at each parent node q_2:

1) If the source tree $\widehat{T}_{q_2}$ associated with q_2 has a node q_1, that is the longest prefix match of $\widehat{p}_1$, let $sub\widehat{T}_{q_2}$ be the q_1-rooted subtree of $\widehat{T}_{q_2}$ and removedVolume be a temporary counter, initialized to 0, for removed volumes. For each prefix p in the list $\mathcal{C}$, let v_p be the volume of p in $sub\widehat{T}_{p_2}$, and:
   - If $sub\widehat{T}_{q_2}$ has a node q that represents the prefix p itself or a parent of p distinct from the root q_1 of

```
1. C is the list of the leaves of sub$\widehat{T}_{p_2}$.
2. Let be $V = \widehat{p}_1$->tVolume the volume of sub$\widehat{T}_{p_2}$.
3. Let be removedVolume a temporary counter.

4. FOR ($q_2 = p_2$->parent; $q_2$!=NULL; $q_2 = q_2$->parent)
5.   IF ($\exists$ sub$\widehat{T}_{q_2} \subset \widehat{T}_{q_2}$, sub$\widehat{T}_{q_2} \supset$ sub$\widehat{T}_{p_2}$)
6.     removedVolume = 0
7.     FOR each prefix p of C,
8.       let $v_p$ be the volume of p in sub$\widehat{T}_{p_2}$
9.       IF sub$\widehat{T}_{q_2}$ has an node q representing p or a
         parent of p distinct from the root of sub$\widehat{T}_{q_2}$
10.        removedVolume+ = $v_p$
11.        delete the descendant of q
12.        discount the volume $v_p$ in $\widehat{T}_{q_2}$
13.      ELSE
14.        $V- = v_p$
15.        remove p from the list C
16.    ENDFOR
17.    remove ($V$ − removedVolume) in $\widehat{T}_{q_2}$
18.  ELSE BREAK
19.ENDFOR
```

TABLE I
PSEUDOCODE FOR THE REMOVAL OF THE LEAVES OF $sub\widehat{T}_{p_2}$.

$sub\widehat{T}_{q_2}$, we delete the descendants of q in the tree $\widehat{T}_{q_2}$ and we remove the volume v_p from $\widehat{T}_{q_2}$, starting from q and traversing the parents of q in postorder; we finally increment removedVolume+ $= v_p$ to count the removed volume.

- Otherwise, we remove the prefix p from the list $\mathcal{C}$. This case means that p has already been removed in an earlier step of the mHHH search algorithm, and it is not needed to remove p from the source trees of parents of q_2. We then remove the value of V from v_p to update the total volume to remove in the upper levels.

When completing the removal of prefixes of the list $\mathcal{C}$ from the source tree $\widehat{T}_{q_2}$, the counter V gives the total volume to remove in $\widehat{T}_{q_2}$ and removedVolume counts the volume that has actually been removed, so that we need to remove (V − removedVolume) in $\widehat{T}_{q_2}$, starting in postorder from the node q_1.

2) Otherwise, we stop here and do not process the remaining parent nodes of p_2, meaning that these nodes do not contain any leaf of $sub\widehat{T}_{p_2}$.

Note that the procedure for the removal of the $\widehat{p}_1$-rooted tree $sub\widehat{T}_{p_2}$ from the source trees of the parent nodes of p_2 ends when one of the following conditions is satisfied:

- we reach a parent node q_2 whose source tree does not contain a node that is the longest prefix match of $\widehat{p}_1$;
- the list $\mathcal{C}$ is getting null (note that when a prefix is removed from the list, the value V decreases of the volume of this prefix).

To establish an analogy with the exhaustive algorithm in Section III, removing the leaves of $sub\widehat{T}_{p_2}$ from the source trees associated with the parents of p_2 is equivalent to replacing the circuits composing the mHHH join (p_1, p_2) with their reduced form. Suppose that p_2 is at level k in the destination tree T_D. The removal of a leaf of $sub\widehat{T}_{p_2}$ from the source trees of the parent nodes of p_2 means that we replace the circuit, modeled by the leaf, with its $(k+1)$-th reduce form. When a leaf of $sub\widehat{T}_{p_2}$ cannot be removed from a local source tree associated with a parent of p_2 at level $k' < k$, this means that the circuit represented by the leaf is already in the $(k'+1)$-th reduced form (in an earlier step of the algorithm), so that we can only reduce the circuit from $(k'+1)$ to $(k+1)$, thus we can only remove the leaf from the source trees associated with the parents of p_2 located at a level between k and k'.

D. Algorithm Analysis

When searching mHHHs, our on-line algorithm operates directly with discounted volumes (by using the circuit based approach) so that we have not to deal with the overcounting problem unlike the algorithm in [6].

Our algorithm satisfies the definition for ε-approximate multidimensional Hierarchical Heavy Hitters, introduced by Cormode *et al.* to assess the accuracy of on-line algorithms. We recall here this definition:

Definition 3: With the notation of Definition 1 in Section II, identifying ϕ-multidimensional hierarchical heavy hitters with an error margin $\varepsilon < \phi$ consists in determining the set P of those nodes p in the lattice and their estimated volumes $\tilde{f}_p$ such that:

1) for $p \in P$, $f_p - \varepsilon N \leq \tilde{f}_p \leq f_p$,
   where $f_p = \sum_{e \in \mathcal{F} \cap S(p)} f_e$ is the true volume of the sublattice rooted at p,
2) for every node $q \notin P$, by setting $S(P) = \bigcup_{p \in P} S(p)$,

$$\sum_{e \in \mathcal{F} \cap (S(q) \setminus S(P))} \tilde{f}_e < \lfloor \phi N \rfloor.$$

The use of the expansion smoother $T_{smooth} = \varepsilon \cdot N/L$ in the update procedure ensures that the first condition of the definition is satisfied, and the second condition is also naturally satisfied since we use the same condition to find mHHHs.

V. EXPERIMENTAL VALIDATION

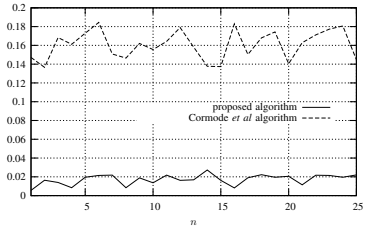
We propose in this section an evaluation of our algorithm using a NetFlow trace [9] from the interconnection IP backbone of France Telecom. The trace is composed of about 60,000 records per second from more than 50 routers and an average of 2,5 million distinct pairs of source and destination addresses per minute.

We evaluate the false positive and negative ratios and the memory consumption of our algorithm while providing a comparison with the algorithm by Cormode *et al.* [6]. We run the algorithms over several consecutive observation periods of duration one minute each. The mHHH threshold is set to $\phi = 0.01$ and the estimation error $\varepsilon = 0.001$. At the end of each observation period, we analyze the output of algorithms: we use our off-line algorithm to compute the actual mHHHs and to assess the accuracy of on-line algorithms.

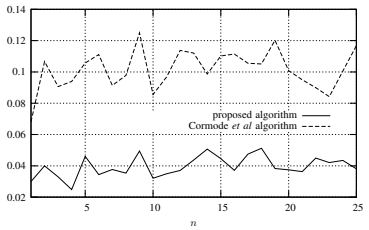
We recall that Cormode *et al.* algorithm uses a lattice-like data structure whose update is an extension of the Lossy Counting algorithm [8], so that there is a compression phase

in their update procedure. The number of compressions during the observation window is a trade-off between memory consumption and the update speed. The more there are compressions within the observation period, the smaller is the memory usage, but the update time may be slowed down: in our evaluation, the average update time per NetFlow record is about 300 μs for Cormode *et al.* algorithm and 20 μs for the proposed one. For memory comparison, we express the memory consumption in number of nodes used by the data structure of each algorithm.

Figures 3(a) and 3(b) display, as a function of the index of the observation window, the false positive and negative ratios. The ratio of false positives is significantly reduced by using our algorithm (about 2 %). The relatively high ratio of false postives in Cormode *et al.* algorithm may be due to the fact that NetFlow data contain very little information on flows because of sampling and that overcounting may frequently occur when rolling up volumes to higher levels in the lattice, while our algorithm uses another approach that allows to cancel the overcounting problem. The false negative ratio of our algorithm is also smaller (less 5 %) as a consequence.



(a) False positive ratio



(b) False negative ratio

Fig. 3. False positive and negative ratios.

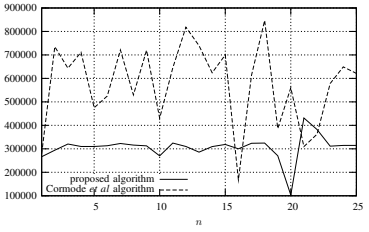


Fig. 4. Memory consumption expressed in number of nodes.

The memory consumption is shown in Figure 4, and for illustration, we display in Figure 5 the number of compressions in the update procedure of Cormode *et al.* algorithm. The number of nodes used by our algorithm is relatively stable, but remains smaller than Cormode *et al.* algorithm.

Note that the off-peak observed at index 20 in Figures 4 and 5 is only due to a decrease of traffic during this observation period (probably a network failure).

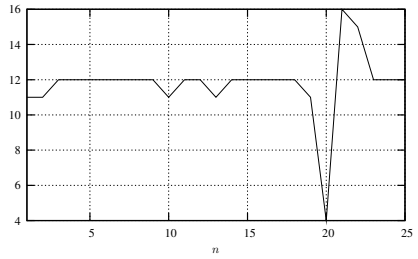


Fig. 5. Number of compressions in the Cormode *et al.* algorithm.

VI. Conclusion

We have presented in this paper a novel approach to the mHHH identification in IP traffic, based on a graph of weighted circuits. The graph models the lattice formed by a stream of source and destination IP address pairs. In this graph, the circuits are IP address pairs and any prefix pair is modeled by a combination of some circuits such that the total weight of these circuits is exactly the discounted volume of the prefix pair, i.e. the volume after discounting the volumes of descendants that are already mHHHs. Using this model allows us to eliminate the overcounting problem when searching mHHH pairs.

We have proposed an off-line algorithm using the circuit based model. From this exhaustive algorithm, we have finally derived an on-line algorithm that achieves efficient accuracy and memory guarantees.

References

[1] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online identification of hierarchical heavy hitters: Algorithms, evaluation and applications," in *Proc. of IMC '04*, October 2004.

[2] P. Truong and F. Guillemin, "Dynamic binary trees for hierachical clustering of IP traffic," in *Proc. of Globecom 2007*, November 2007.

[3] Y. Lin and H. Liu, "Separator: sifting hierarchical heavy hitters accurately from data streams," in *Proc. of ADMA '07*, August 2007.

[4] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Finding hierarchical heavy hitters in data streams," in *Proc. of VLDB*, September 2003.

[5] C. Estan, S. Savage, and G. Varghese, "Automatically inferring patterns of resource consumption in network traffic," in *Proceedings of ACM SIGCOMM*, 2003.

[6] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data," in *Proc. of ACM SIGMOD*, June 2004.

[7] J. Hershberger, N. Shrivastava, S. Suri, and C. D. Tth, "Space complexity of hierarchical heavy hitters in multi-dimensional data streams," in *Proc. of ACM PODS '05*, June 2005.

[8] G. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proc. of VLDB*, August 2002.

[9] "Introduction to Cisco IOS NetFlow," http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/prod_white_paper0900aecd80406232.pdf.