MVPipe: Enabling Lightweight Updates and Fast Convergence in Hierarchical Heavy Hitter Detection (Supplementary File)

Lu Tang, Qun Huang, Patrick P. C. Lee

In this supplementary file, we present (i) the major operations of 2D HHH detection, (ii) the proofs of theorems for 2D HHH detection, and (iii) the additional experimental results for IPv6 traffic.

I. 2D HHH Detection

We elaborate on the major operations of 2D HHH detection, whose pseudo-code is shown in Figure 1.

The Update operation builds on the Push function, which inserts the input (key, value) starting from the array at node (a_0,b_0) and stops until it reaches the array at node (a_1,b_1) or key is admitted by a candidate HHH. The Update operation is invoked for each incoming packet (f,v_f). Initially, the fields of each bucket of MVPipe are zeros. Upon the arrival of (f,v_f), we call the function $\mathsf{Push}(f,v_f,(0,0),(d-1,d-1))$ to process f from level 0 of the lattice until it reaches the top node or is admitted by a candidate HHH at one of the nodes in the lattice.

The Detect operation is invoked at the end of each epoch. It addresses the double counting problem via the ConCount function, which returns the estimated conditioned count of a key x respect to $\mathcal{H}$ based on $\hat{S}(x)$, the coordinate (a,b) of x in the lattice, and $\mathcal{H}$. The ConCount function calculates the estimated conditioned count of key x based on the inclusion-exclusion principle. Specifically, let $\mathcal{P}$ be the set of the closest descendants of x in $\mathcal{H}$ (Line 38). To obtain the conditioned count of x without the double-counting errors, we first subtract the count $L(y)$ of each key y in $\mathcal{P}$ from the estimated conditioned count of x (Lines 39-44). We show that the count $L(y)$ is the lower bound of $S(y)$ in the proof of Theorem 5. Then for each pair of distinct keys in $\mathcal{P}$, we obtain the closest common descendant of the two keys, where the count of the common descendant is doubly deducted from x (Line 46). We add back the count of each common descendant to x to resolve the double counting error (Lines 47-51).

II. Proofs

We present the proofs of the theorems for 2D HHH detection.

A. Proof of Theorem 4

Proof. The space usage and detection time in 2D HHH detection can be derived as in Theorem 1. The per-packet update in 2D HHH detection accesses at most d buckets along the source direction and d buckets along the destination direction, thereby taking $O(d)$ time. □

B. Proof of Theorem 5

Proof. Our proof focuses on the coverage property only, as the proof for the accuracy property follows that in Theorem 2. We first consider $S_{\mathcal{H}}(x)$. Let $\mathcal{P}$ be the set of the closest descendants of x in $\mathcal{H}$. Let $\mathcal{Q}$ be the set of the common closest descendants of any two distinct keys y and y' in $\mathcal{P}$, that have no other ancestors in $\mathcal{P}$ except y and y'. By the definition of the conditioned count and the inclusion-exclusion principle [2], we have $S_{\mathcal{H}}(x) = S(x) - \sum_{y\in\mathcal{P}} S(y) + \sum_{z\in\mathcal{Q}} S(z)$.

We show that $\hat{S}_{\mathcal{H}}(x)$ given by MVPipe is always larger than $S_{\mathcal{H}}(x)$. From the ConCount and Estimate functions, we have $\hat{S}_{\mathcal{H}}(x) = \hat{S}(x) - \sum_{y\in\mathcal{P}} L(y) + \sum_{z\in\mathcal{Q}} \hat{S}(z) \geq S_{\mathcal{H}}(x)$, where $L(y)$ is the sum of the cumulative count of y and the cumulative counts of y's descendants in $\mathcal{H}$. By Lemma 1 and the accuracy property, we have $\hat{S(x)} \geq S(x)$, $L(y) \leq S(y)$, and $\hat{S}(z) \geq S(z)$. Then, $\hat{S}_{\mathcal{H}}(x) \geq S_{\mathcal{H}}(x)$.

Thus, x is not reported as an HHH only if it is not in $K_{i,h_i(x)}$ (suppose x is hashed to $B(i,j)$). We then push the count of x to higher-level nodes until it is admitted by an HHH. That is, at least one of x's ancestors is in $\mathcal{H}$. By the definition of the conditioned count, $S_{\mathcal{H}}(x) = 0$, which leads to a contradiction. □

III. Additional Experiments on IPv6 Traffic

In §VI of the main paper, we show that MVPipe achieves high accuracy and high update throughput for IPv4 traffic. In this section, we compare MVPipe with the state-of-the-art schemes listed in §VI of the main paper for IPv6 traffic. We aim to show that MVPipe maintains its accuracy and performance gains for IPv6 traffic as well. Also, our observations for IPv4 traffic regarding the accuracy and performance trends for all schemes still hold for IPv6 traffic.

A. Traces

We consider two different sources of traces in our evaluation: (i) the same CAIDA traces as in §VI of the main paper and (ii) the traces from MAWI's WIDE project [1]. For the CAIDA traces, we filter out the IPv4 traffic and focus on the IPv6 traffic only, where each epoch (of length one minute) contains 0.42 M packets and 5.9 K unique IPv6 source addresses on average. Since the last 64 bits of each IPv6 address are zeros in the CAIDA traces, we only consider the first 64 bits of an IPv6 address in our evaluation. For the MAWI traces, we consider the traces at samplepoint-C [1], in which the daily IPv6 traffic was captured on an IPv6 line connected to 6Bone in January

```
1: function PUSH(key, value, $(a_0,b_0),(a_1,b_1)$)
2:     $(x,v_x) \leftarrow$ (key, value)
3:     for $p = a_0$ to $a_1$ do
4:         $(y,v_y) \leftarrow (x,v_x)$
5:         for $q = b_0$ to $b_1$ do
6:             $y \leftarrow$ generalize $y$ to node $(p,q)$
7:             $i \leftarrow p \times d + q$    ▷ $i$ is the index of the corresponding array
8:             $V_{i,h_i(y)} \leftarrow V_{i,h_i(y)} + v_y$
9:             if $K_{i,h_i(y)} = y$ then
10:                $I_{i,h_i(y)} \leftarrow I_{i,h_i(y)} + v_y$
11:                $C_{i,h_i(y)} \leftarrow C_{i,h_i(y)} + v_y$
12:                if $q = 0$ then
13:                    return
14:                end if
15:                $b_0 \leftarrow 0$
16:                break
17:            else if $I_{i,h_i(y)} \geq v_y$ then
18:                $I_{i,h_i(y)} \leftarrow I_{i,h_i(y)} - v_y$
19:            else
20:                $I_{i,h_i(y)} \leftarrow v_y - I_{i,h_i(y)}$
21:                if $q = 0$ then
22:                    $(x,v_x) \leftarrow (K_{i,h_i(y)},C_{i,h_i(y)})$
23:                end if
24:                swap $(K_{i,h_i(y)},C_{i,h_i(y)})$ and $(y,v_y)$
25:            end if
26:        end for
27:        $b_0 \leftarrow 0$
28:    end for
29: end function
30: function ESTIMATE($x,(a,b),t$)
31:    $v \leftarrow 0$
32:    $b' \leftarrow \min\{b+t,d-1\}$
33:    for $q = b$ to $b'$ do
34:        $i \leftarrow a \times d + q$
35:        $y \leftarrow$ generalize $y$ to node $(a,q)$
36:        if $K_{i,h_i(x)} = y$ then
37:            $U_q = (V_{i,h_i(y)} + I_{i,h_i(y)})/2 + v$
38:            $v \leftarrow v + C_{i,h_i(y)}$
39:        else
40:            $U_q = (V_{i,h_i(y)} - I_{i,h_i(y)})/2 + v$
41:        end if
42:    end for
43:    return $\min_{b \leq q \leq b'}\{U_q\}$
44: end function
45: function CONCOUNT($x,\hat{S}(x),(a,b),\mathcal{H}$)
46:    $\hat{S}_{\mathcal{H}}(x) \leftarrow \hat{S}(x)$
47:    $\mathcal{P} \leftarrow \{(y,(p,q)) \in \mathcal{H} \mid \nexists y' \in \mathcal{H} : y \prec y' \prec x\}$
48:    for each $(y,(p,q)) \in \mathcal{P}$ do
49:        $i \leftarrow p \times d + q$
50:        $\mathcal{T} \leftarrow \{(y',(p',q')) \in \mathcal{H}, (y' \prec y)—p' = p$ or $q' = 0\}$
51:        $i' \leftarrow p' \times d + q'$
52:        $L(y) = C_{i,h_i(y)} + \sum_{y' \in \mathcal{T}} C_{i',h_{i'}(y')}$
53:        $\hat{S}_{\mathcal{H}}(x) \leftarrow \hat{S}_{\mathcal{H}}(x) - L(y)$
54:    end for
55:    for each pair of distinct element $y,y'$ in $\mathcal{P}$ do
56:        $(y'',(p'',q'')) \leftarrow$ closest common descendant of $y$ and $y'$
57:        if $\nexists z \neq y,y' \in \mathcal{P}$, s.t. $y'' \prec z$ then
58:            $\mathcal{T} \leftarrow \{(z',(p',q')) \in \mathcal{H}, (z' \prec z)—p' = p$ or $q' = 0\}$
59:            $i' \leftarrow p' \times d + q'$
60:            $\hat{S}(z) =$ ESTIMATE$(z,(p,q),t) + \sum_{z' \in \mathcal{T}} C_{i',h_{i'}(z')}$
61:            $\hat{S}_{\mathcal{H}}(x) \leftarrow \hat{S}_{\mathcal{H}}(x) + \hat{S}(z)$
62:        end if
63:    end for
64:    return $\hat{S}_{\mathcal{H}}(x)$
65: end function
66: procedure UPDATE($x,v_x$)
67:    PUSH$(x,v_x,(0,0),(d-1,d-1))$
68: end procedure
69: procedure DETECT
70:    $\mathcal{H} \leftarrow \emptyset$
71:    for $p = 0$ to $d-1$ do
72:        for $q = 0$ to $d-1$ do
73:            $i \leftarrow p \times d + q$
74:            for $j = 0$ to $w_i - 1$ do
75:                $x \leftarrow K_{i,j}$
76:                $U(x) \leftarrow$ ESTIMATE$(x,(p,q),t)$
77:                $\mathcal{T} \leftarrow \{(x',(p',q')) \in \mathcal{H}, (x' \prec x)—p' = p$ or $q' = 0\}$
78:                $i' \leftarrow p' \times d + q'$
79:                $\hat{S}(x) \leftarrow U(x) + \sum_{x' \in \mathcal{T}} C_{i',h_{i'}(x')}$
80:                $\hat{S}_{\mathcal{H}}(x) \leftarrow$ CONCOUNT$(x,\hat{S}(x),(p,q),\mathcal{H})$
81:                if $\hat{S}_{\mathcal{H}}(x) \geq \phi\mathcal{S}$ then
82:                    $\mathcal{H} \leftarrow \mathcal{H} \cup (x,\hat{S}(x))$
83:                else
84:                    if $q = 0$ then
85:                        PUSH$(x,C_{i,j},(p,q+1),(d-1,d-1))$
86:                    else
87:                        PUSH$(x,C_{i,j},(p,q+1),(p,d-1))$
88:                    end if
89:                end if
90:            end for
91:        end for
92:    end for
93:    return $\mathcal{H}$
94: end procedure
```

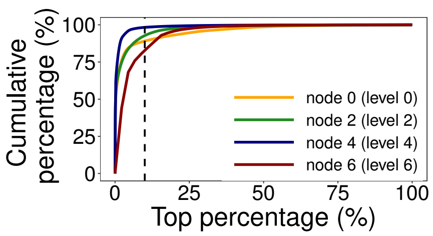
Figure 1: Major operations in 2D HHH detection.

2008. While the traces were collected more than a decade ago, we believe that the MAWI traces are sufficiently representative for two reasons: (i) the 6Bone network is a dedicated testbed purely for IPv6 traffic and is not mixed with IPv4 traffic as in the CAIDA traces and (ii) the traces show similar skewness as observed in IPv4 traffic (see details below). In our evaluation, we set the epoch length of the MAWI traces as one day to include sufficient amounts of traffic, in which each epoch contains 2 M packets and 3.98 K unique IPv6 addresses on average. By default, we keep the same parameter settings as in §VI of the main paper.

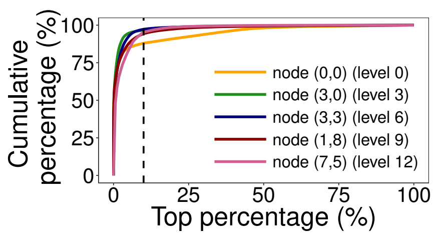
B. Skewness Analysis

Before we start our experiments, we first analyze whether the skewness property we observed in §III of the main paper still holds across the aggregation levels for IPv6 traffic. Figure 2 plots the cumulative percentage of packet counts versus the top percentage of keys for the 1D-byte and 2D-byte hierarchies across different aggregation levels for both CAIDA and MAWI traces. We observe that the top-10% of keys at each level all account for more than 80% and 91% of IP traffic for the level in the CAIDA and MAWI traces, respectively. Thus, we show that the skewness of IPv6 traffic is also observed.

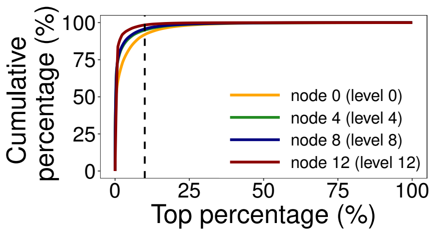
(Experiment S1) Accuracy comparisons for IPv6 traffic. We compare different HHH detection schemes on accuracy versus different values of the absolute threshold. For the CAIDA



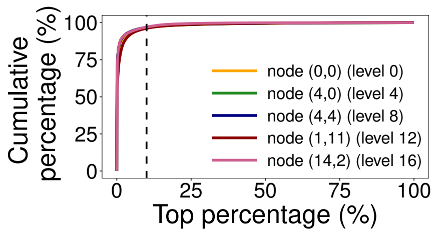
(a) 1D-byte hierarchy, CAIDA



(b) 2D-byte hierarchy, CAIDA

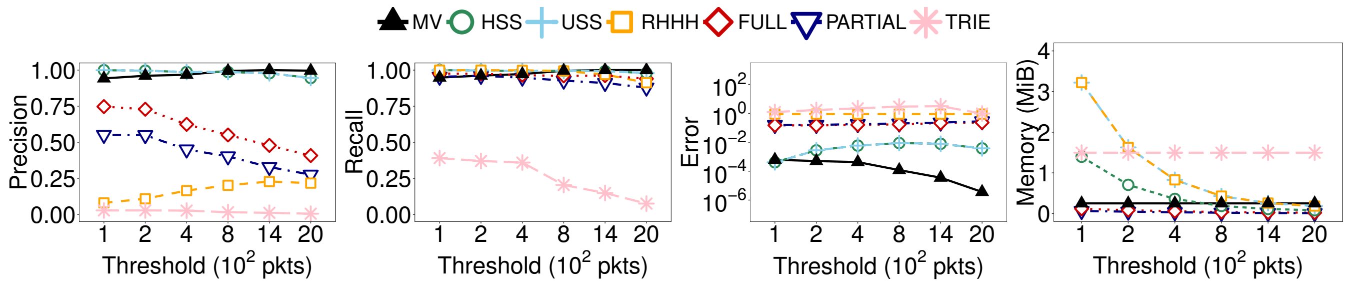


(c) 1D-byte hierarchy, MAWI

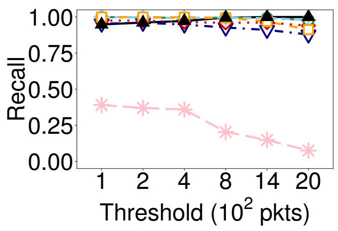


(d) 2D-byte hierarchy, MAWI

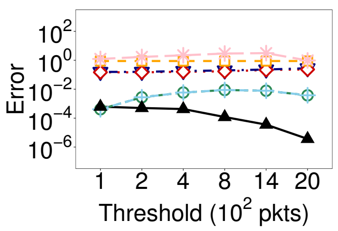
Figure 2: Cumulative percentage of packet counts versus the top-percentage of keys at different aggregation levels for IPv6 traffic in CAIDA (figures (a)-(b)) and MAWI (figures (c)-(d)). The dashed line denotes the top-10% mark.



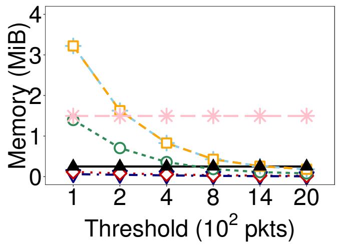
(a) Precision for 1D-byte



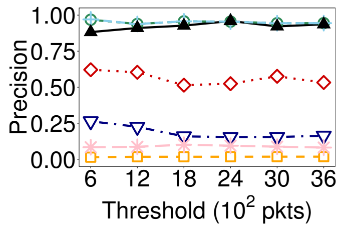
(b) Recall for 1D-byte



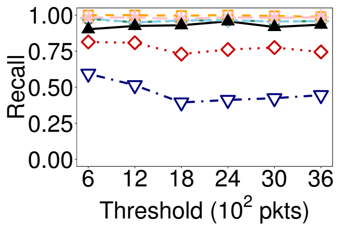
(c) Error for 1D-byte



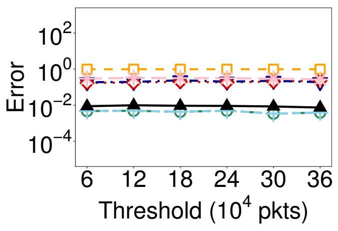
(d) Memory for 1D-byte



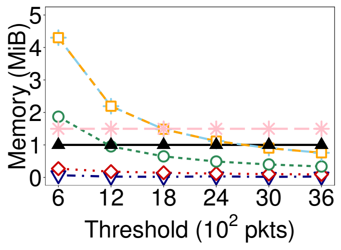
(e) Precision for 1D-bit



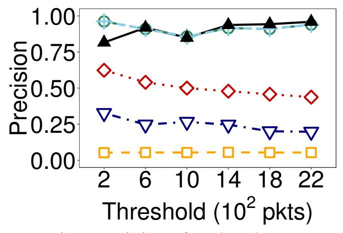
(f) Recall for 1D-bit



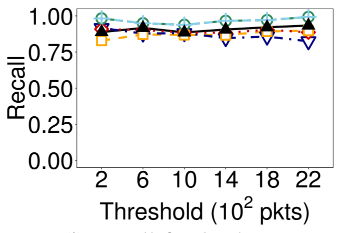
(g) Error for 1D-bit



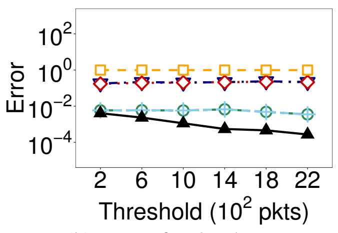
(h) Memory for 1D-bit



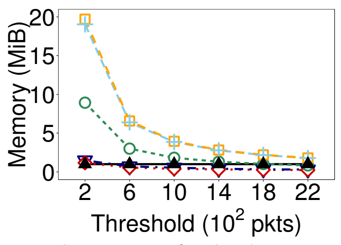
(i) Precision for 2D-byte



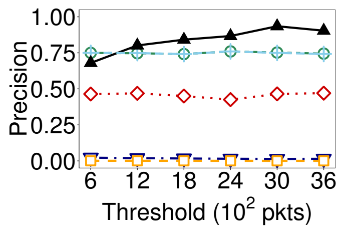
(j) Recall for 2D-byte



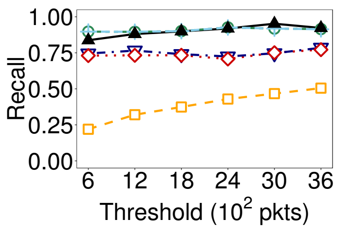
(k) Error for 2D-byte



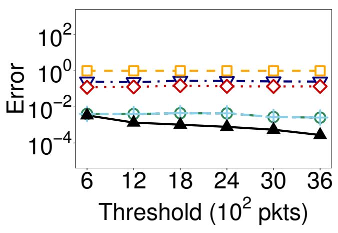
(l) Memory for 2D-byte



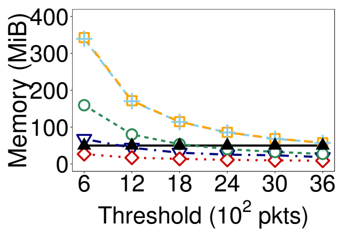
(m) Precision for 2D-bit



(n) Recall for 2D-bit

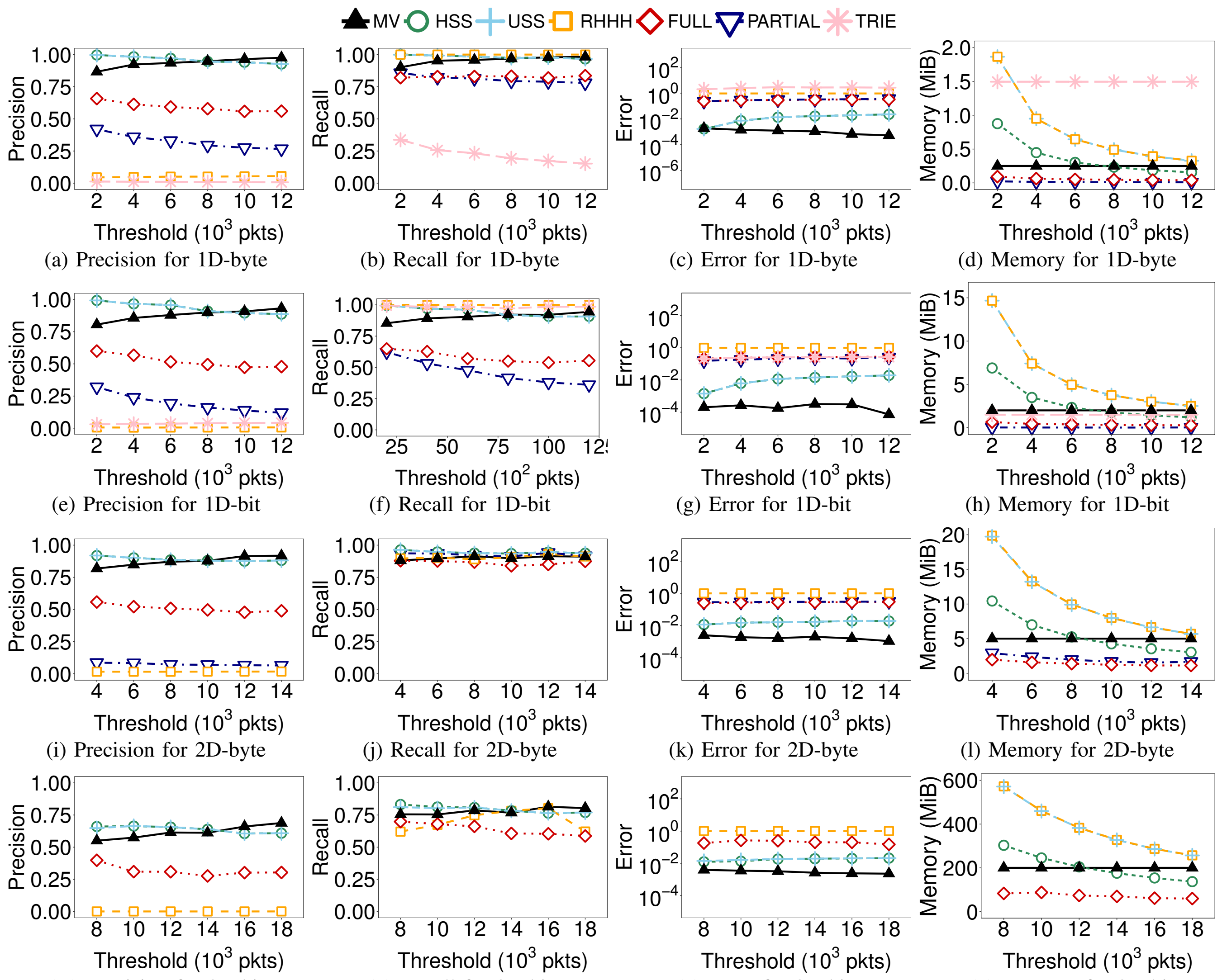


(o) Error for 2D-bit



(p) Memory for 2D-bit

Figure 3: (Experiment S1) Accuracy comparisons for IPv6 traffic in CAIDA.



(a) Precision for 1D-byte (b) Recall for 1D-byte (c) Error for 1D-byte (d) Memory for 1D-byte

(e) Precision for 1D-bit (f) Recall for 1D-bit (g) Error for 1D-bit (h) Memory for 1D-bit

(i) Precision for 2D-byte (j) Recall for 2D-byte (k) Error for 2D-byte (l) Memory for 2D-byte

(m) Precision for 2D-bit (n) Recall for 2D-bit (o) Error for 2D-bit (p) Memory for 2D-bit

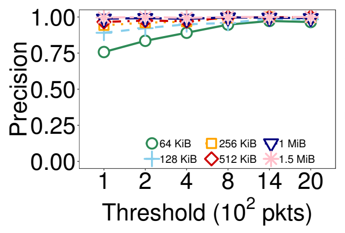
Figure 4: (Experiment S1) Accuracy comparisons for IPv6 traffic in MAWI.

traces, we fix the memory space of MVPipe as 256 KiB, 1 MiB, 1.5 MiB, and 90 MiB for 1D-byte, 1D-bit, 2D-byte, and 2D-bit HHH detection, respectively. For the MAWI traces, we fix the memory space of MVPipe as 256 KiB, 2 MiB, 5 MiB, and 200 MiB for 1D-byte, 1D-bit, 2D-byte, and 2D-bit HHH detection, respectively. We consider the same accuracy metrics as in Experiment 1 of the main paper. Figures 3 and 4 show the results. Since PARTIAL cannot return any results for 2D-bit HHH detection on the MAWI traces after running more than 36 hours, we do not plot its results in Figures 4(m)-4(p). We observe that all schemes show a similar trend as in Experiment 1. MVPipe achieves high accuracy and medium size of memory usage among all cases.

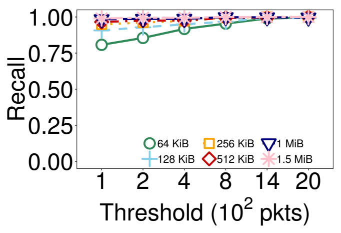
(Experiment S2) Robustness of MVPipe for IPv6 traffic. We evaluate MVPipe versus the absolute threshold by varying the memory size allocated for MVPipe. Figures 5 and 6 show the results. The accuracy of MVPipe remains robust for IPv6 traffic.

(Experiment S3) Update throughput for IPv6 traffic. We benchmark the update throughput of all schemes with the same setting as in Experiment 3. Figure 7 shows the results in the CAIDA traces. MVPipe achieves the highest throughput with up to 6.36× and 40.31× throughput gain for byte-level and bit-level HHH detection, respectively. We observe similar results on the MAWI traces as shown in Figure 8, where the throughput gains for byte-level and bit-level HHH detection are up to 9.61× and 69.72×, respectively. Note that the throughput of MVPipe is higher than that in IPv4 traffic. The reason is that the skewness of IPv6 traffic is higher than that of IPv4 traffic, so MVPipe only needs to access fewer arrays and hence achieves faster processing.

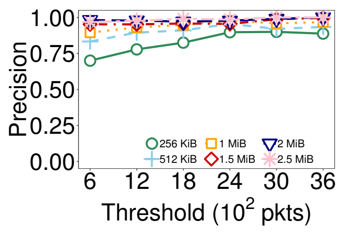
(Experiment S4) Throughput versus skewness. We benchmark the update throughput of all HHH detection schemes on a server equipped with an Intel i7-11700 2.50GHz CPU and 16GiB RAM. The server runs Ubuntu 20.04. We vary the skewness degree of the IPv6 traffic in both the CAIDA and MAWI traces by using the method described in Experiment 5. In the original IPv6 traffic, the top-100 flows account for 73% and 75% of the total number of packets in the CAIDA and MAWI traces, respectively. Figures 9 and 10 show the results.



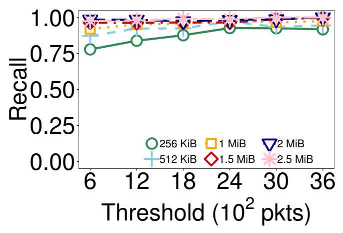
(a) Precision for 1D-byte



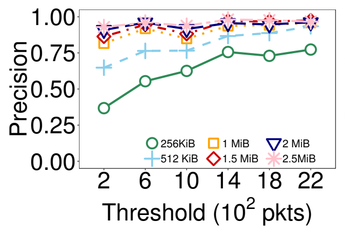
(b) Recall for 1D-byte



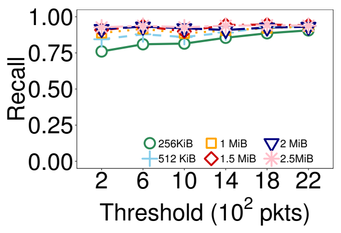
(c) Precision for 1D-bit



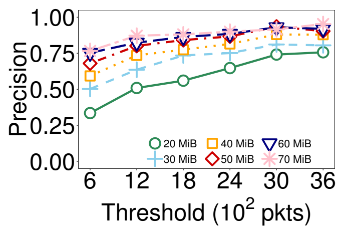
(d) Recall for 1D-bit



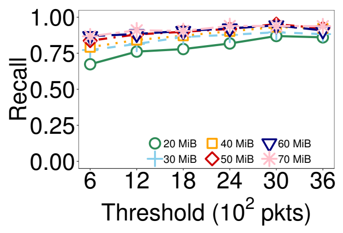
(e) Precision for 2D-byte



(f) Recall for 2D-byte

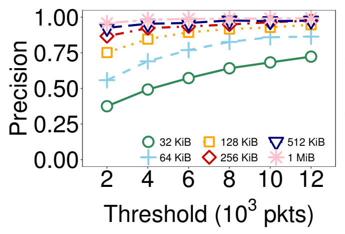


(f) Precision for 2D-bit

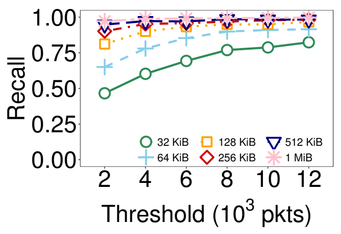


(h) Recall for 2D-bit

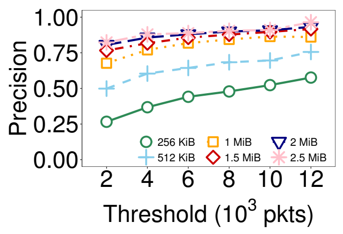
Figure 5: (Experiment S2) Robustness of MVPipe for IPv6 traffic in CAIDA.



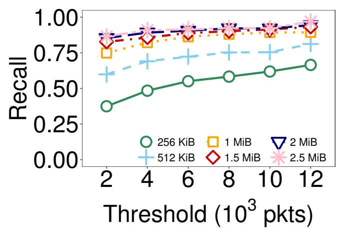
(a) Precision for 1D-byte



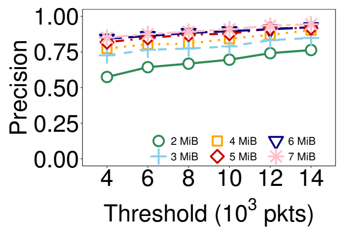
(b) Recall for 1D-byte



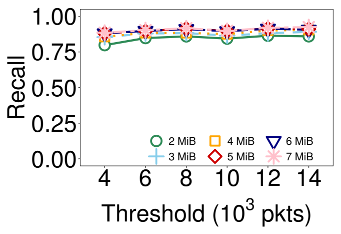
(c) Precision for 1D-bit



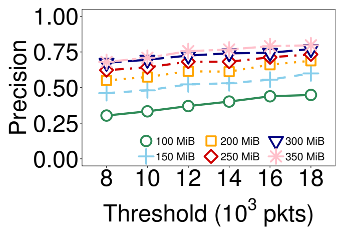
(d) Recall for 1D-bit



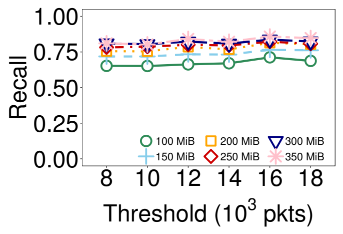
(e) Precision for 2D-byte



(f) Recall for 2D-byte

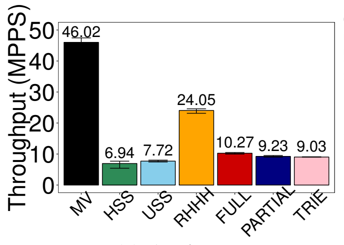


(f) Precision for 2D-bit

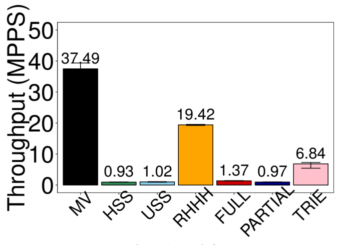


(h) Recall for 2D-bit

Figure 6: (Experiment S2) Robustness of MVPipe for IPv6 traffic in MAWI.

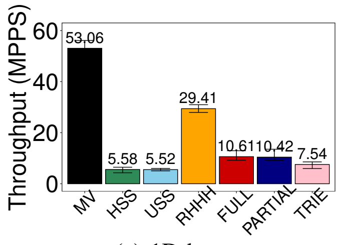


(a) 1D-byte

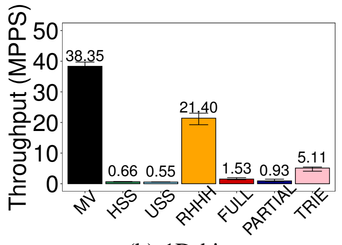


(b) 1D-bit

Figure 7: (Experiment S3) Update throughput for IPv6 traffic in CAIDA.



(a) 1D-byte



(b) 1D-bit

Figure 8: (Experiment S3) Update throughput for IPv6 traffic in MAWI.

MVPipe shows a similar update performance as for IPv4 traffic. The throughput of MVPipe decreases for less skewed traces, yet it remains higher than other schemes except for RHHH and TRIE.

(Experiment S5) Number of traversed nodes for IPv6 traffic. Figures 11 and 12 show the cumulative percentage of packets versus the number of traversed nodes by MVPipe for different skewness degrees. For the original IPv6 traffic in the CAIDA traces (i.e, the black curve), in 1D-byte HHH detection, 92% of packet updates traverse only one node in the hierarchy, where each packet updates on average traverses only 1.09 nodes. In 1D-bit HHH detection, 87% of packet updates traverse one node, while each packet update on average traverses 1.21 nodes. For the MAWI traces, 89% of packet updates traverse only one node in 1D-byte HHH detection, where each packet update on average traverses only 1.1242 nodes. Also, 90% of packet

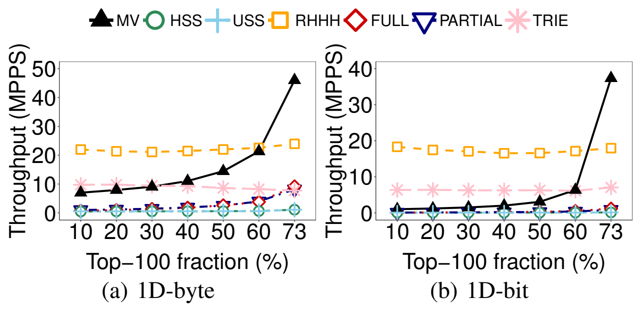


Figure 9: (Experiment S4) Throughput vs. skewness in CAIDA.

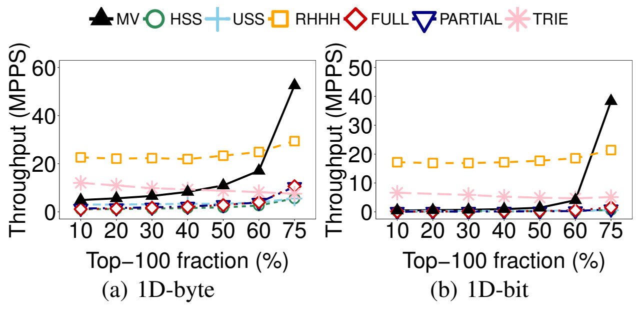


Figure 10: (Experiment S4) Throughput vs. skewness in MAWI.

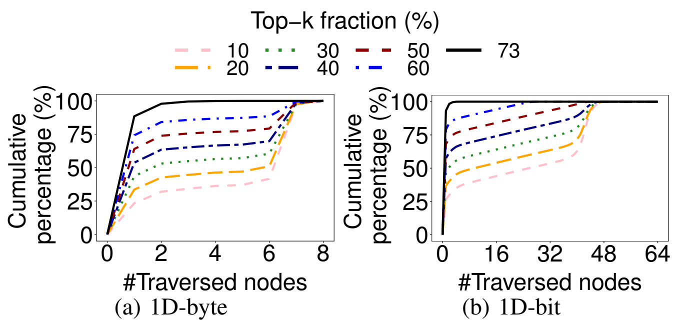


Figure 11: (Experiment S5) Number of traversed nodes for IPv6 traffic in CAIDA.

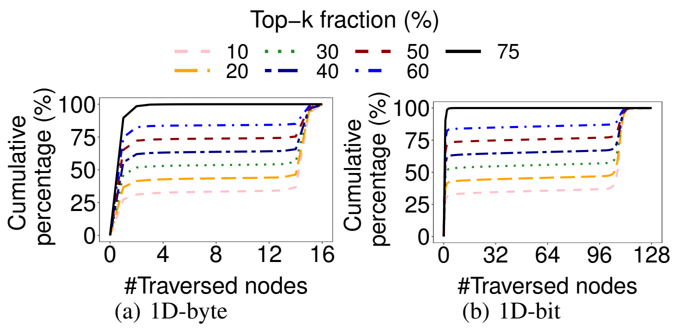


Figure 12: (Experiment S5) Number of traversed nodes for IPv6 traffic in MAWI.

updates traverse one node in 1D-bit HHH detection, where each packet update on average traverses only 1.1201 nodes. As the skewness degree decreases, the number of traversed nodes for each packet update increases.

References

[1] K. Cho, K. Mitsuya, and A. Kato. Traffic Data Repository at the WIDE Project. In *USENIX 2000 FREENIX Track*, June 2000. https://mawi.wide.ad.jp/mawi/.

[2] M. Mitzenmacher, T. Steinke, and J. Thaler. Hierarchical Heavy Hitters with the Space Saving Algorithm. In *Proc. of ACM ALENEX*, pages 160–174, 2012.