Appendix A
Missing Proof in Section IV

A. Proof of Theorem 4

Proof. For an arbitrary edge (x, y) (different from (u, v)), We define three indicative variables

$$I^{(1)}_{u,v,x,y,j} = \begin{cases} 1, & \text{if } x = u \wedge y \neq v \wedge c_j(y) = c_j(v); \\ 0, & \text{otherwise;} \end{cases}$$

$$I^{(2)}_{u,v,x,y,j} = \begin{cases} 1, & \text{if } x \neq u \wedge y = v \wedge r_j(x) = r_j(u); \\ 0, & \text{otherwise;} \end{cases}$$

$$I^{(3)}_{u,v,x,y,j} = \begin{cases} 1, & \text{if } x \neq u \wedge y \neq v \wedge (r_j(x), c_j(y)) = (r_j(u), c_j(v)); \\ 0, & \text{otherwise,} \end{cases}$$

Due to the independence of hash functions, we know that $EI^{(1)}_{u,v,x,y,j} = \frac{1}{n}$ if $u = x$, $EI^{(2)}_{u,v,x,y,j} = \frac{1}{n}$ if $v = y$, and $EI^{(3}_{u,v,x,y,j} = \frac{1}{n^2}$ otherwise. Let us define another variable

$$X_{u,v,j} = \sum_y f(u,y) I^{(1)}_{u,v,u,y,j} + \sum_x f(x,v) I^{(2)}_{u,v,x,v,j} + \sum_{x \neq u \wedge y \neq v} f(u,v) I^{(3)}_{u,v,x,y,j}$$

indicating the overestimation error caused by hash collision. Due to the linearity of expectation,

$$EX_{u,v,j} = \sum_y f(u,y) EI^{(1)}_{u,v,u,y,j} + \sum_x f(x,v) EI^{(2)}_{u,v,x,v,j} + \sum_{x \neq u \wedge y \neq v} f(u,v) EI^{(3)}_{u,v,x,y,j} = \frac{N_s}{n} + \frac{N_d}{n^2}.$$

Note that $\bar{f}(u, v) = f(u, v) + \min\{X_{u,v,j} : 1 \leq j \leq s\}$. According to the Markov Inequality, we get

$$P(\bar{f}(u,v) \geq f(u,v) + \varepsilon) = P(X_{u,v,j} \geq \varepsilon, \forall 1 \leq j \leq s)$$
$$= [P(X_{u,v,j} \geq \varepsilon)]^s \leq \left(\frac{EX_{u,v,j}}{\varepsilon}\right)^s = \left(\frac{N_s}{\varepsilon n} + \frac{N_d}{\varepsilon n^2}\right)^s. \quad \square$$

B. Proof of Theorem 5

Proof. The key idea is: every incoming item with edge (u, v) will increment either P or T, and P will only be incremented when the edge matches the key in Stage 1. We will prove the theorem based on mathematical induction. Let $\underline{f}_t(u, v) = P_t, f_t(u, v), \bar{f}_t(u, v) = P_t + T_t$ be three values after inserting t^{th} item in the graph stream. Initially we have $\underline{f}_0(u, v), f_0(u, v), \bar{f}_0(u, v) = 0$. Suppose the $(t + 1)^{th}$ item in the graph stream is $e_{t+1} = (u_{t+1}, v_{t+1}, w_{t+1})$. There are several cases:

Case 1: (u, v) is recorded in Stage 1.

1.1) If $(u_{t+1}, v_{t+1}) = (u, v)$, then $P_{t+1} = P_t + w_{t+1}$, and $T_{t+1} = T_t$, so all three values will be incremented by w_{t+1}, and $\underline{f}_{t+1}(u, v) \leq f_{t+1}(u, v) \leq \bar{f}_{t+1}(u, v)$.

1.2) If $(u_{t+1}, v_{t+1}) \neq (u, v)$, and does not evict (u, v) in Stage 1, then $P_{t+1} = P_t$ and $T_{t+1} \geq T_t$, so we still have $\underline{f}_{t+1}(u, v) \leq f_{t+1}(u, v) \leq \bar{f}_{t+1}(u, v)$.

1.3) If $(u_{t+1}, v_{t+1}) \neq (u, v)$, and evict (u, v) in Stage 1, then (u, v) will be inserted into Stage 2. In this situation, we have $P_{t+1} = 0, T_{t+1} = P_t + T_t$, so $0 = \underline{f}_{t+1}(u, v) \leq f_{t+1}(u, v) = f_t(u, v) \leq \bar{f}_t(u, v) = P_t + T_t = P_{t+1} + T_{t+1} = \bar{f}_{t+1}(u, v)$.

Case 2: (u, v) is not recorded in Stage 1.

2.1) If $(u_{t+1}, v_{t+1}) = (u, v)$, then either $P_{t+1} = w_t$ (if (u, v) enters Stage 1), or $T_{t+1} = T_t + w_t$ (if (u, v) does not enter Stage 1). In both situation, we have $\underline{f}_{t+1}(u, v) \leq f_{t+1}(u, v) \leq \bar{f}_{t+1}(u, v)$.

2.2) If $(u_{t+1}, v_{t+1}) \neq (u, v)$, then $P_{t+1} = 0$, and $T_{t+1} \geq T_t$, so $\underline{f}_{t+1}(u, v) \leq f_{t+1}(u, v) \leq \bar{f}_{t+1}(u, v)$.

In all situations we have $\underline{f}_{t+1}(u, v) \leq f_{t+1}(u, v) \leq \bar{f}_{t+1}(u, v)$. By mathematical induction we know that Theorem 5 holds. $\square$

C. Proof of Corollary 6

Proof. Just replace N_s with N'_s, and N_d with N'_d in Theorem 4, then we get the result. $\square$

D. Proof of Theorem 7

Proof. The proof is similar to Theorem 4. Similarly define $I^{(1)}_{u,v,x,y,j}, I^{(2)}_{u,v,x,y,j}, I^{(3)}_{u,v,x,y,j}$ and $X_{u,v,j}$, then $EI^{(1)}_{u,v,x,y,j} = \frac{1}{n_j}$ if $u = x$, $EI^{(2)}_{u,v,x,y,j} = \frac{1}{n_j}$ if $v = y$, and $EI^{(3}_{u,v,x,y,j} = \frac{1}{n_j^2}$ otherwise. Hence $EX_{u,v,j} = \frac{N_s}{n_j} + \frac{N_d}{n_j^2}$. According to the Markov inequality, we get

$$P(\bar{f}(u,v) \geq f(u,v) + \varepsilon) = P(X_{u,v,j} \geq \varepsilon, \forall t \leq j \leq s)$$
$$= \prod_{j=t}^{s} P(X_{u,v,j} \geq \varepsilon) \leq \prod_{j=t}^{s} \frac{EX_{u,v,j}}{\varepsilon} = \prod_{j=t}^{s} \left(\frac{N_s}{\varepsilon n_j} + \frac{N_d}{\varepsilon n_j^2}\right). \quad \square$$

E. Proof of Theorem 9

Proof. The proof is similar to Theorem 5. The key idea is: every incoming item with edge (u, v) will increment P, T or F, and P will only be incremented when the edge matches the key in Stage 1. In addition, the sum of P, T and F will not decrease. Based on mathematical induction, Theorem 9 holds. $\square$

F. Proof of Theorem 10

Proof. The proof is similar to Corollary 6 and 8. The key idea is, if (u, v) enters Stage 1 and is frozen in Error Funnel, then its counter in the highest level of Stage 2 will not increment due to hash collision, so the error only arises from hash collision before (u, v) enters Stage 1. Hence both equation holds. $\square$

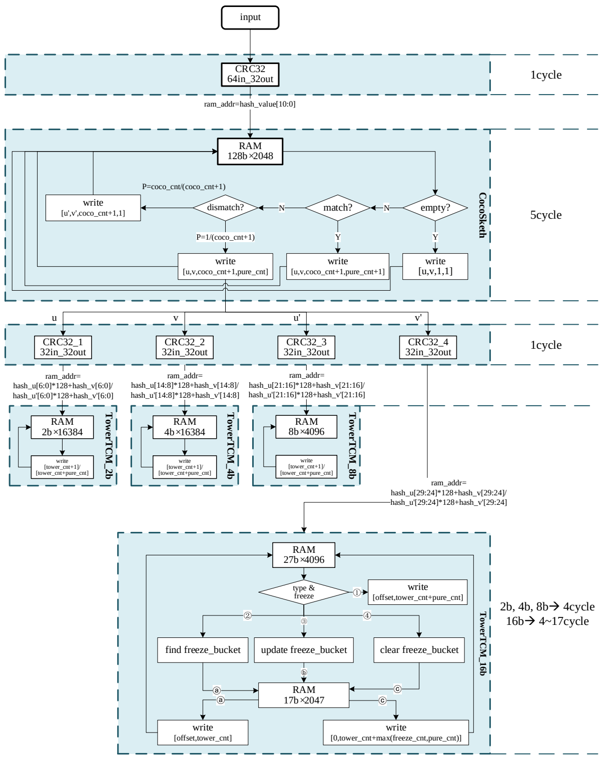
G. Proof of Theorem 11

Proof. Let $t' = \log t$. Since every 2^{k+l-1} adjacent counters in B_s are grouped together in Level l of Error Funnel, the index field in the freezing bucket only needs to record the relative address of the frozen counter, which uses $k+l-1$ bits. Hence, the total memory cost of Error Funnel is

$$\sum_{l=1}^{t'-k+1} (k + l - 1 + \delta'_s) \cdot 2^{t'+1-k-l}$$
$$= (k + \delta'_s) \sum_{l=1}^{t'-k+1} 2^{t'+1-k-l} + \sum_{l=1}^{t'-k+1} (l-1) \cdot 2^{t'+1-k-l}$$
$$= (k + \delta'_s)(2^{t'-k+1} - 1) + 2^{t'-k+1} + k - t' - 2$$
$$\leq (k + \delta'_s + 1) \cdot 2^{\log t - k+1}. \quad \square$$

Module	Resource Overhead			Frequency (MHz)
	LUTs	Register	Block RAM Tile	
Hash	463 (0.11%)	279 (0.03%)	0	290
CocoSketch	665 (0.15%)	760 (0.09%)	7.5 (0.51%)	290
TowerTCMSketch	947(0.22%)	2019 (0.23%)	16 (1.09%)	290
Total	2075 (0.48%)	3058 (0.35%)	23.5 (1.60%)	290

TABLE II: Performance on FPGA Platform.



input
CRC32 64in_32out
1cycle
ram_addr=hash_value[10:0]
RAM 128b×2048
P=coco_cnt/(coco_cnt+1)
write [u',v',coco_cnt+1,1]
dismatch?
N
match?
N
empty?
Y
Y
P=1/(coco_cnt+1)
write [u,v,coco_cnt+1,pure_cnt]
write [u,v,coco_cnt+1,pure_cnt+1]
write [u,v,1,1]
CocoSketh
5cycle
u
v
u'
v'
CRC32_1 32in_32out
CRC32_2 32in_32out
CRC32_3 32in_32out
CRC32_4 32in_32out
1cycle
ram_addr= hash_u[6:0]*128+hash_v[6:0]/ hash_u'[6:0]*128+hash_v'[6:0]
ram_addr= hash_u[14:8]*128+hash_v[14:8]/ hash_u'[14:8]*128+hash_v'[14:8]
ram_addr= hash_u[21:16]*128+hash_v[21:16]/ hash_u'[21:16]*128+hash_v'[21:16]
RAM 2b×16384
RAM 4b×16384
RAM 8b×4096
write [tower_cnt+1]/ [tower_cnt+pure_cnt]
TowerTCM_2b
TowerTCM_4b
TowerTCM_8b
ram_addr= hash_u[29:24]*128+hash_v[29:24]/ hash_u'[29:24]*128+hash_v'[29:24]
RAM 27b×4096
type & freeze
①
write [offset,tower_cnt+pure_cnt]
②
③
④
find freeze_bucket
update freeze_bucket
clear freeze_bucket
ⓐ
ⓑ
ⓒ
RAM 17b×2047
write [offset,tower_cnt]
write [0,tower_cnt+max(freeze_cnt,pure_cnt)]
TowerTCM_16b
2b, 4b, 8b→ 4cycle
16b→ 4~17cycle

Fig. 23: FPGA Implementation Architecture

H. Proof of Theorem 12

Proof. To insert an edge (u, v) with weight w, HourglassSketch first checks it in d buckets in Stage 1. If (u, v) is already in one of these buckets, we just increment the coco counter and pure counter by w and return, which costs $O(d)$ time; otherwise, we perform probability replacement in Stage 1: if (u, v) replaces an edge (u', v') in Stage 1, we try to freeze (u, v) and unfreeze (u', v') in Error Funnel and insert (u, v) into Stage 2. Since Error Funnel has $\log n_s^2 - k + 1$ levels, freezing and unfreezing operation can be done in $O(\log n_s^2 - k + 1) = O(\log n_s - k)$ time, and insertion operation in Stage 2 costs $O(s)$ time; if (u, v) fails to replace an edge in Stage 1, we insert (u, v) into Error Funnel or Stage 2, which can be done in $O(\log n_s - k + s)$. Hence the insertion time complexity of HourglassSketch is at most $O(d + \log n_s - k + s)$. □

Appendix B
Hardware Implementation

In this section, we implement HourglassSketch on two hardware platforms, *i.e.* FPGA and P4 tofinos.

A. FPGA Implementation

We implement HourglassSketch on an FPGA network experimental platform (Virtex-7). The FPGA integrated with the platform is xc7vx690tffg1761-2 with 433200 Slice LUTs, 866400 Slice Register, and 1470 Block RAM Tile. To make HourglassSketch hardware-friendly, we set $d = 1$ to remove circular dependency in Stage 1. The implementation mainly consists of three hardware modules: calculating hash values (Hash), writing an item into Stage 1 (CocoSketch), and writing an item into Stage 2 (TowerTCMSketch). FPGA-based HourglassSketch is fully pipelined, which can input one element group in every clock. Stage 1 writing is completed after six clocks, and Stage 2 writing is completed after eleven clocks.

Table II shows the clock frequency and all hardware resources used by HourglassSketch.The clock frequency of our implementation in FPGA is 290 MHz, meaning that the throughput of the system can be 290 Mops.

B. P4 Implementation

We have fully built a P4 prototype of HourglassSketch on the Tofino switch [65]. The Tofino switch processes packets in a pipeline manner, so we set $d = 1$. Also, both freezing and unfreezing operation take place on Error Funnel when replacement occurs in Stage 1, so Tofino switch has to access Error Funnel twice to perform two operations. We list the utilization of various hardware resources on the switch in Table III when $m = 1024, s = 4, \delta_1 = 2, \delta_2 = 4, \delta_3 = 8, \delta_4 = 16, n_1 = n_2 = 128, n_3 = n_4 = 64$ and $k = 4$.

TABLE III: Hardware Resources Used by HourglassSketch

Resource	Usage	Percentage
Hash bits	875	17.53%
Exact Xbar	241	15.69%
VLIW Instr	33	8.59%
Stateful ALU	0	0
SRAM	32	3.33%
Map RAM	32	5.56%

C. Discussion

Regarding hardware-friendliness, two key factors determine whether a sketch is suitable for FPGA or P4 implementation:

- Simplicity of the algorithm: Hardware platforms demand algorithms with minimal complexity to achieve high-speed processing and efficient resource usage.
- Unidirectional working logic: For optimal hardware efficiency, algorithms operating in a unidirectional manner are always preferred, allowing items to pass sequentially through the structure without backtracking.

The data structure of HourglassSketch is simple: its components are built upon three hardware-friendly sketches, *i.e.* CocoSketch, TowerSketch and TCMSketch; Besides, the working logic of Error Funnel follows a strict unidirectional flow: items pass from the first level to the last level with no backtracking involved. Consequently, HourglassSketch is highly suitable for FPGA and P4 tofinos. In contrast, GSS and Auxo have more complex operations and lack unidirectional logic, making Auxo hard to implement on hardware. While GSS can run on FPGA, its insertion throughput is limited to 1.7 Mops [33], significantly lower than HourglassSketch's 290 Mops.

APPENDIX C
PSEUDO-CODE OF HOURGLASSSKETCH

A. *Pseudo-code of HourglassSketch Insertion Procedure*

The pseudo-code of insertion procedure of the basic version of HourglassSketch is shown in Algorithm 1.

Algorithm 1: HourglassSketch Insertion Procedure

```
Input: an edge $(u, v)$ with weight $w$
1  for $1 \le i \le d$ do
2      if $A_i[h_i(u,v)].key = (u,v)$ then
3          // Key matching
4          $A_i[h_i(u,v)].pure \leftarrow A_i[h_i(u,v)].pure + w$;
5          $A_i[h_i(u,v)].coco \leftarrow A_i[h_i(u,v)].coco + w$;
6          return;
7  $k \leftarrow \arg\min_{1 \le i \le d} A_i[h_i(u,v)].coco$;
8  $(u', v') \leftarrow A_k[h_k(u,v)].key$;
9  $A_k[h_k(u,v)].coco \leftarrow A_k[h_k(u,v)].coco + w$;
10 $(u,v)$ replaces $(u',v')$ w.p. $\frac{w}{A_k[h_k(u,v)].coco}$;
11 if $(u,v)$ replaces $(u',v')$ then
12     // Insert $(u',v')$ into Stage 2
13     for $1 \le j \le s$ do
14         $B_j[r_j(u')][c_j(v')] \leftarrow$
               $B_j[r_j(u')][c_j(v')] + A_k[h_k(u,v)].pure$;
15     $A_k[h_k(u,v)].pure \leftarrow w$;
16 else
17     // Insert $(u,v)$ into Stage 2
18     for $1 \le j \le s$ do
19         $B_j[r_j(u)][c_j(v)] \leftarrow B_j[r_j(u)][c_j(v)] + w$;
```

B. *Pseudo-code of Error Funnel*

The pseudo-code of freezing and unfreezing operation of Error Funnel is shown in Algorithm 2 and Algorithm 3.

Algorithm 2: Error Funnel Freezing Operation

```
Input: an edge $(u, v)$
1  for $1 \le l \le \log t - k + 1$ do
2      find the group $B_s[r_s(u)][c_s(v)]$ belongs to in level
         $l$;
3      if the index field of the freezing bucket in the group
         is $(r_s(u), c_s(v))$ then
4          // already frozen
5          return;
6      if the freezing bucket in the group is empty then
7          // empty freezing bucket
8          set its index field to $(r_s(u), c_s(v))$;
9          set its counter field to 0;
10         return;
```

Algorithm 3: Error Funnel Unfreezing Operation

```
Input: an edge $(u', v')$ and its pure counter $pure$
1  $sum \leftarrow 0$;
2  for $1 \le l \le \log t - k + 1$ do
3      find the group $B_s[r_s(u')][c_s(v')]$ belongs to in
         level $l$;
4      if the index field of the freezing bucket in the group
         is $(r_s(u'), c_s(v'))$ then
5          increment $sum$ by its counter field;
6          clear the bucket;
7  $B_s[r_s(u')][c_s(v')] \leftarrow$
     $B_s[r_s(u')][c_s(v')] + \max\{sum, pure\}$;
```