Efficient Measurement on Programmable Switches Using Probabilistic Recirculation

Xiaoqi Chen, Ran Ben Basat,
Gil Einziger, Ori Rottenstreich

https://arxiv.org/abs/1808.03412
(to appear in ICNP'18)

NOKIA Bell Labs
PRINCETON UNIVERSITY
Technion Israel Institute of Technology

P4 enables novel monitoring applications

- **"In-Band Network Telemetry" (INT)**
- **High Speed: Filtering and computation at line-rate**
- **Customized sketch data structure**

Why Do Heavy-Hitter Detection?

Better resource allocation:
10% elephant flows consumes 90% resources?

Security:
Unexpected heavy-hitter flow could be an attack

Immediate Action in Data Plane:

Can we re-route the heaviest flows? Drop them? ECN them?

The "heavy-hitter detection" problem (online frequency estimation)

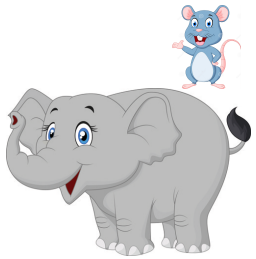
You only see one packet at a time… with flow ID

Count the size of each flow.

Two problem settings:

1. Report elephants at the end of the day (Top-K)

2. For each packet, report a count (RMSE)



Consider data-plane-only algorithm, not involving Controllers (e.g. FlowRadar)

As a streaming computation problem, you never have enough memory.

- 100Gbps line rate, 10s of MB memory

Sample and Hold

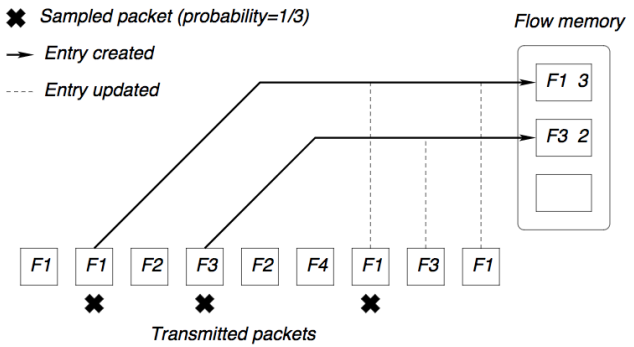
"Early" algorithm (Estan 2003 survey)

Sample packet w.p. *p*

- If counting bytes, sample packet w.p. ***p*size***

If sampled, add a counter to flow table entr

- (If matched this flow ID, increment counter)



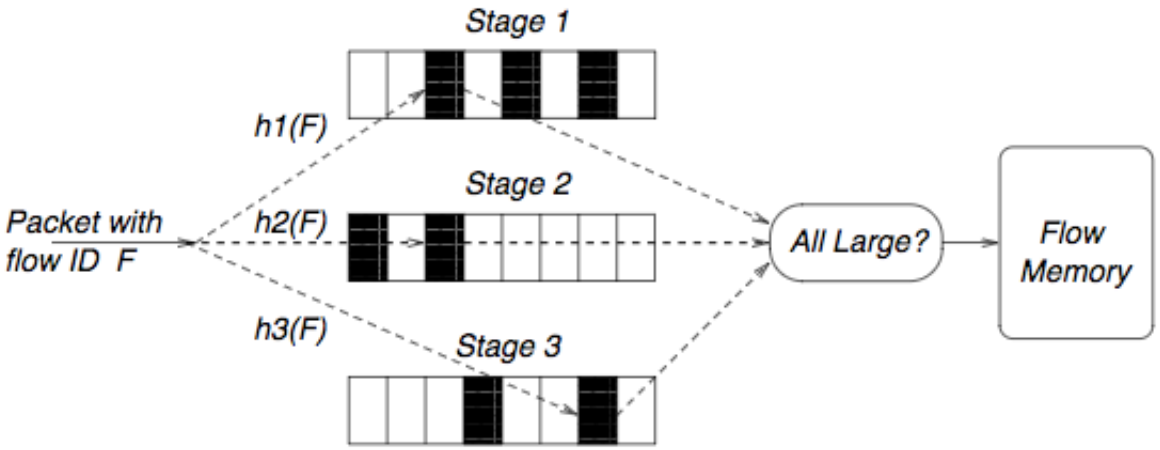
Theoretical promise: "With 99.99% succes

Problem: to catch top 100, you may need 1000 table entries.

The parallel multistage filter

(a.k.a. Count-Min sketch/Counting Bloom Filter)

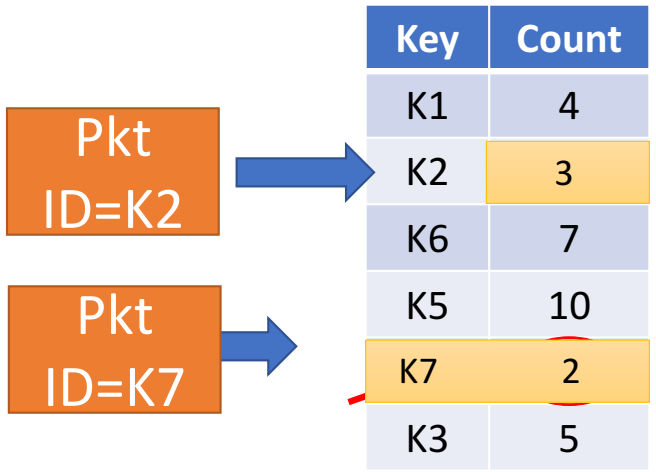
Also early algorithm; reduce false positive, no false negative



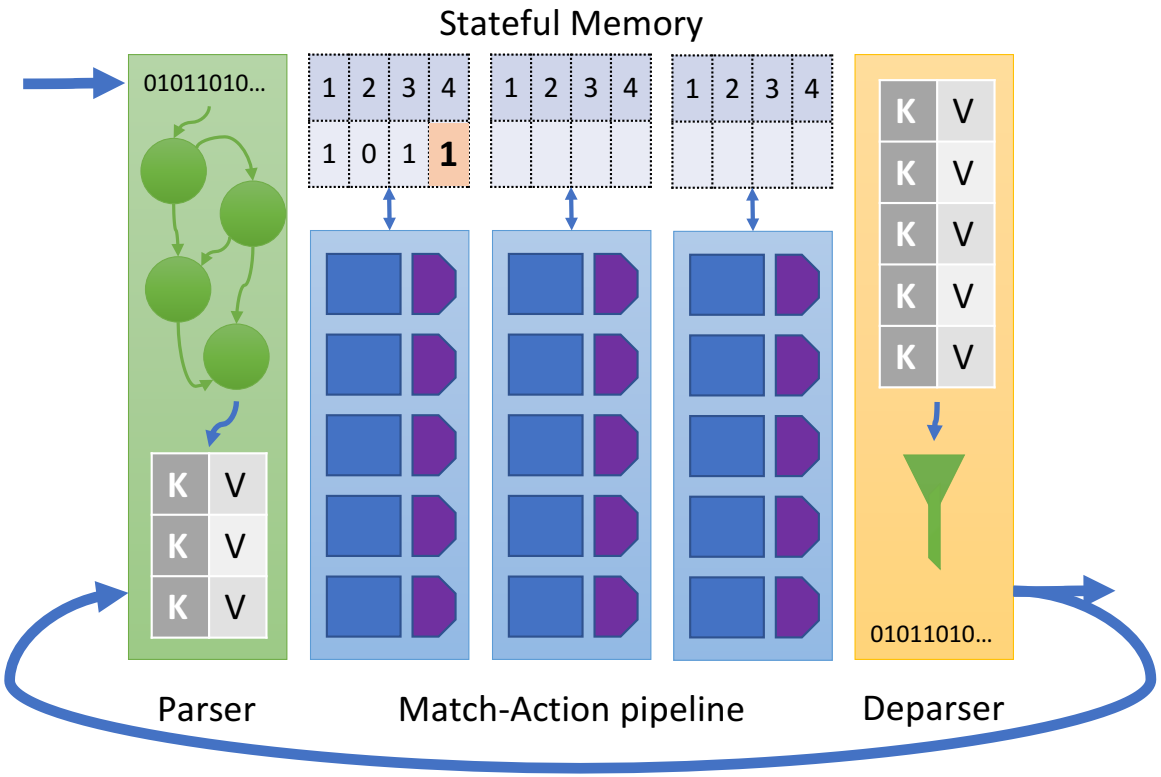
The Space-Saving sketch

If in table, increase counter

Otherwise, replace minimum item, and increase counter



PISA/RMT architectural constraints

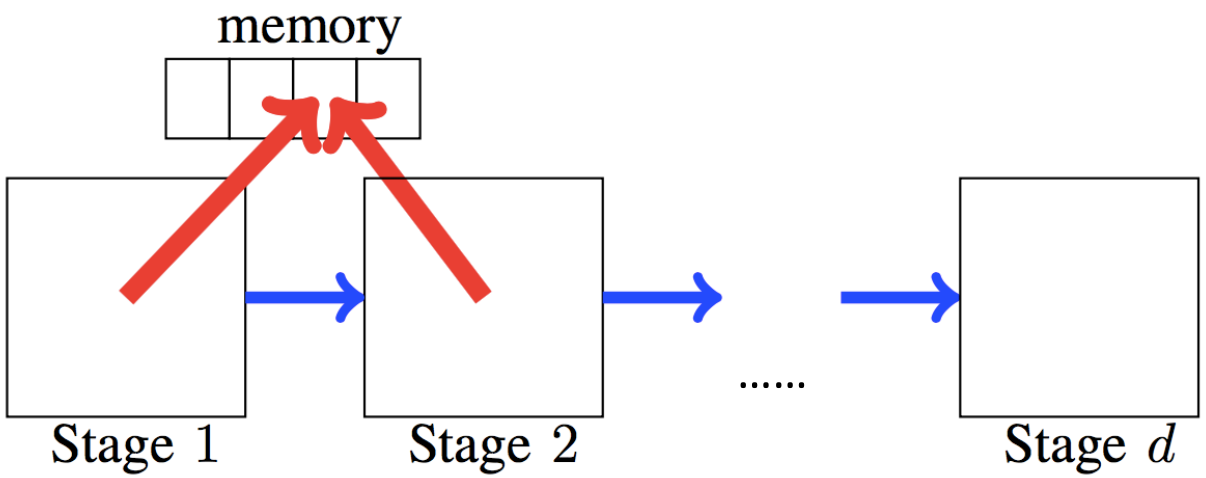


Constraint: No Memory Access Across Stages

Each stage processes a different packet simultaneously.

If both stages write to the same location, a memory hazard will occur.

Thus, access to each writable memory block is limited to one stage.

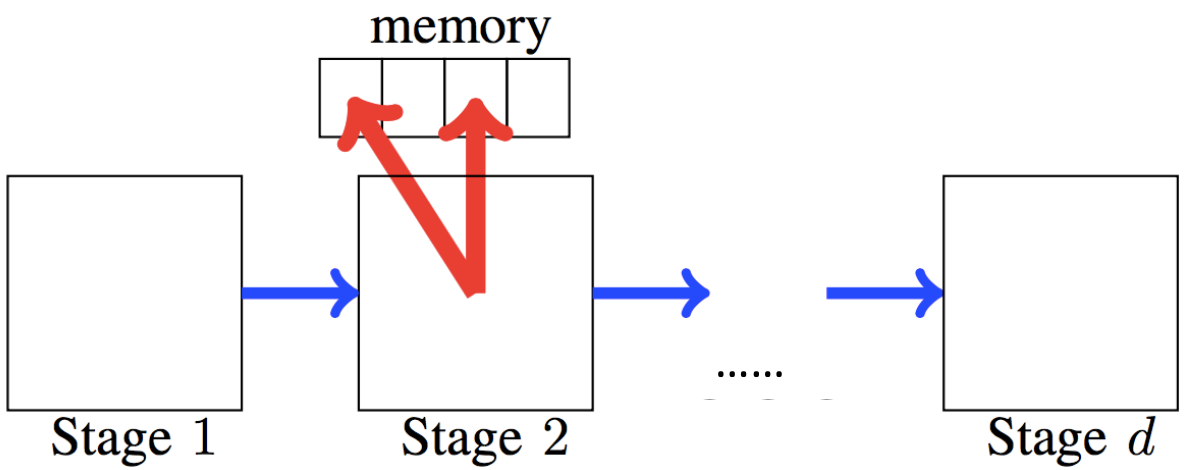


Constraint: Read One Location Per Stage

To deal with line-rate, need constant-time access to memory. (cannot use content-addressable ram/CAM)

Can only specify one SRAM address, then read/write from it.

Cannot access multiple locations

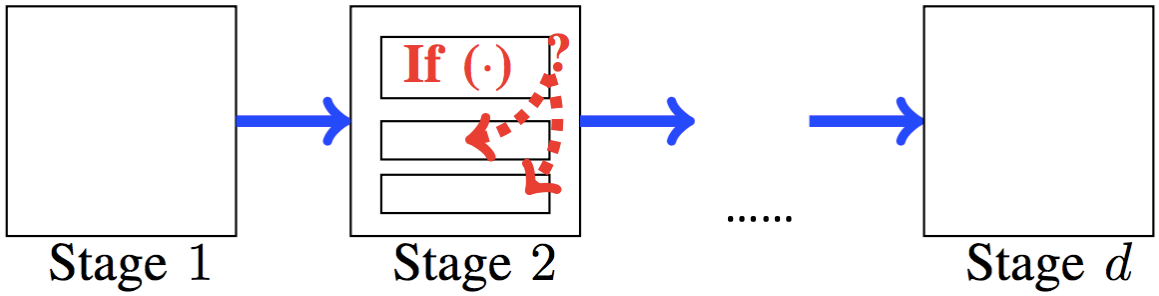


Constraint: Limited Branching Within Stage

Action Unit within stages are simple ALUs, does not support for complex branching

Branching is cheap when happen between stages, via Match-Action Table

Limited branching during Stateful Memory access



Constraints: Limited #memory/#stages

Not a big deal for approximate algorithm designer. (We'll see why later.)

How to adapt Space-Saving to PISA switches?

A programmable switch saw a packet. Now it can access register memory.

Limitation: you can only access x register elements.

Cannot compute minimum! Space-Saving cannot work!

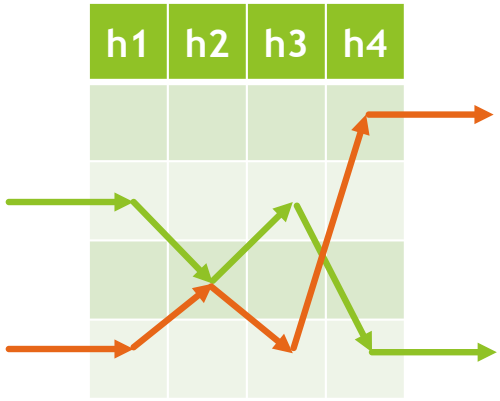
- PISA model is more restrictive than streaming computation / online algorithm

Dirty fix 1: compute minimum out of 4 counters.

- Good, but what if 5 ~~pigeons~~ elephants falls into 4 holes?

Dirty fix 2: independent hash functions, like CBF

- OK, now it works



HashPipe [Sivaraman SOSR'17]:

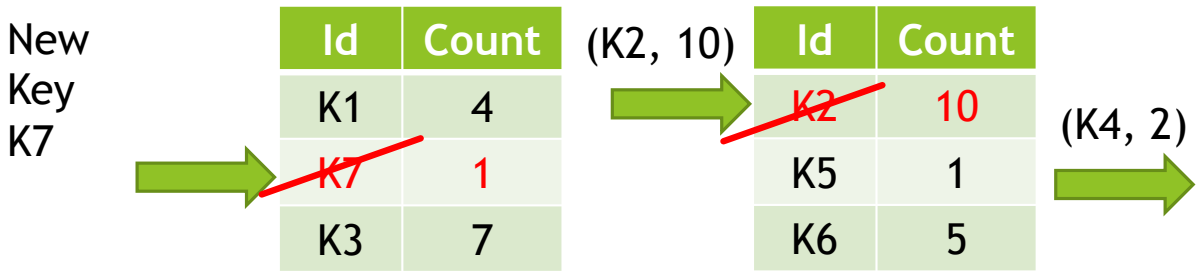
Hardware-friendly, implement x-way minimum using x stages

"Carry" the minimum value across the pipeline.

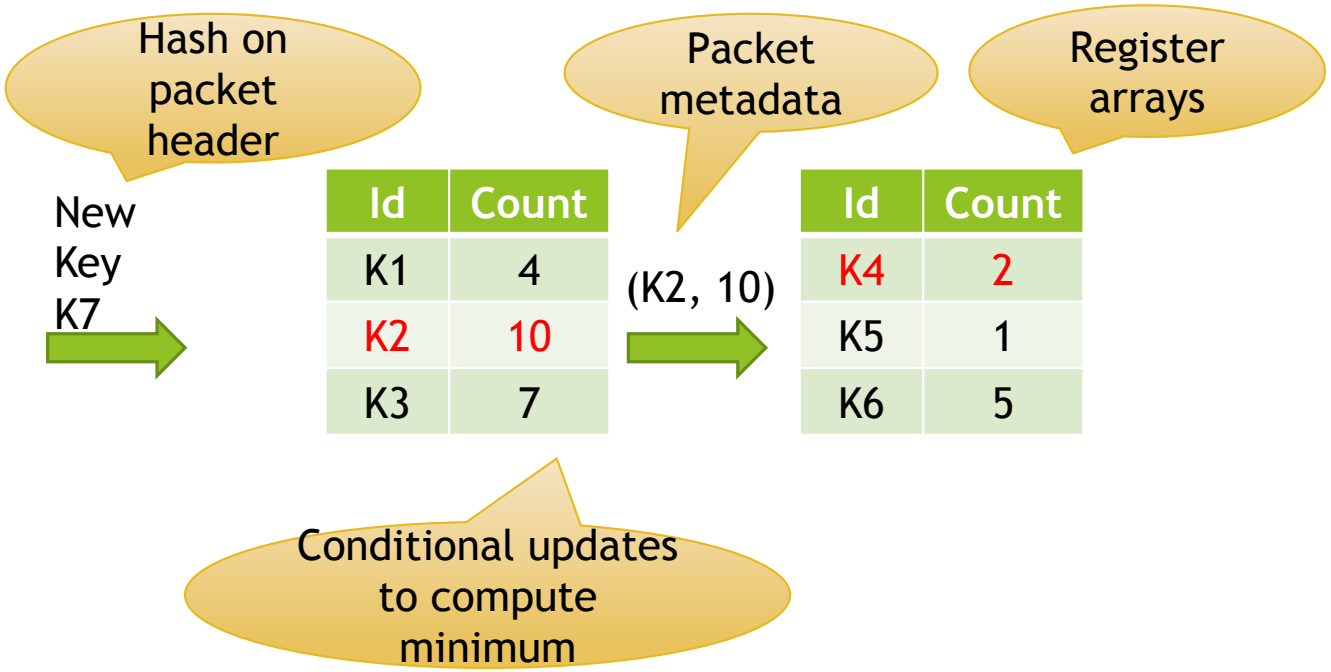
- (Always insert at first stage.)

Acceptable performance loss (compared with Space-Saving), better than Sample and Hold

2-stage example:



HashPipe is hardware-friendly for PISA



Problem of HashPipe

1. Duplicate entries

HashPipe always insert to 1st stage. Thus, mice flows still always evicts others.
(Perform poorly on heavy-tail workload)

2. Stagnant accuracy-memory tradeoff

Given more memory, HashPipe did not improve much

3. Performance Gap from Space-Saving

Better algorithm for Heavy-Hitter

Always evicting minimum is a bad idea from Space-Saving

- One mouse kills the k-th elephant
- Thrashing, if n+1 elephants in n holes
- Probabilistic eviction? What probability?

Intuition: even if a packet is not remembered, we want to report the correct number (in expectation)!

- Solution: we probabilistically replace the minimum entry.
- W.p. ***1-p*** we do not replace, output 0
- W.p. ***p*** we replace and increment, output c_min+1
- In expectation, we ”increment the count by 1” => ***p=1/(c_min+1).***
- ***The RAP algorighm***

PRECISION: defer decision to the end

Packet K7

Flow Key	Counter
K1	15
K3	27
K4	33
K2	12
K8	46

Flow Key	Counter
K5	12
K7	4
K19	6
K6	22
K11	25

Flow Key	Counter
K32	16
K45	7
K36	2
K31	42
K17	37

1/(3+1)=25%
recirc probability

The PRECISION algorithm: Probabilistic Recirculation Admission

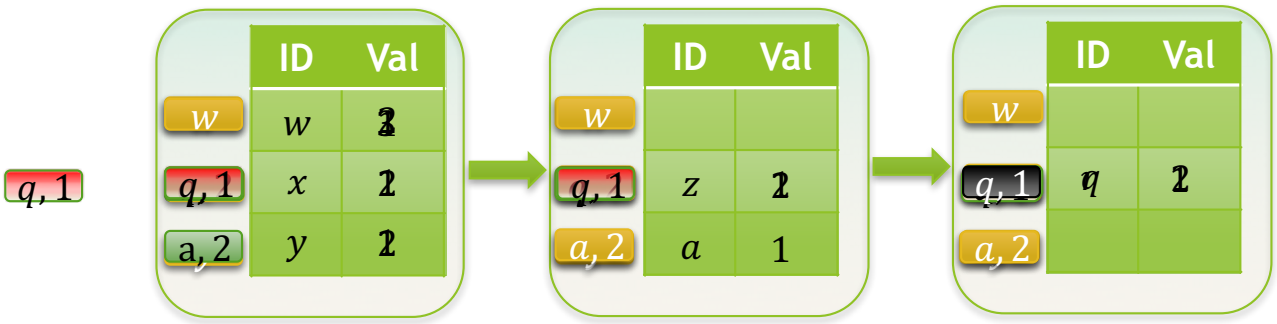
Key idea:

- Find the minimum across x entries (one entry per stage)
- If matched in some stage, bump counter (and leave)
- Otherwise recirculate packet w.p. ***1/(c_min+1)***
- When recirculate, replace the minimum entry's ID

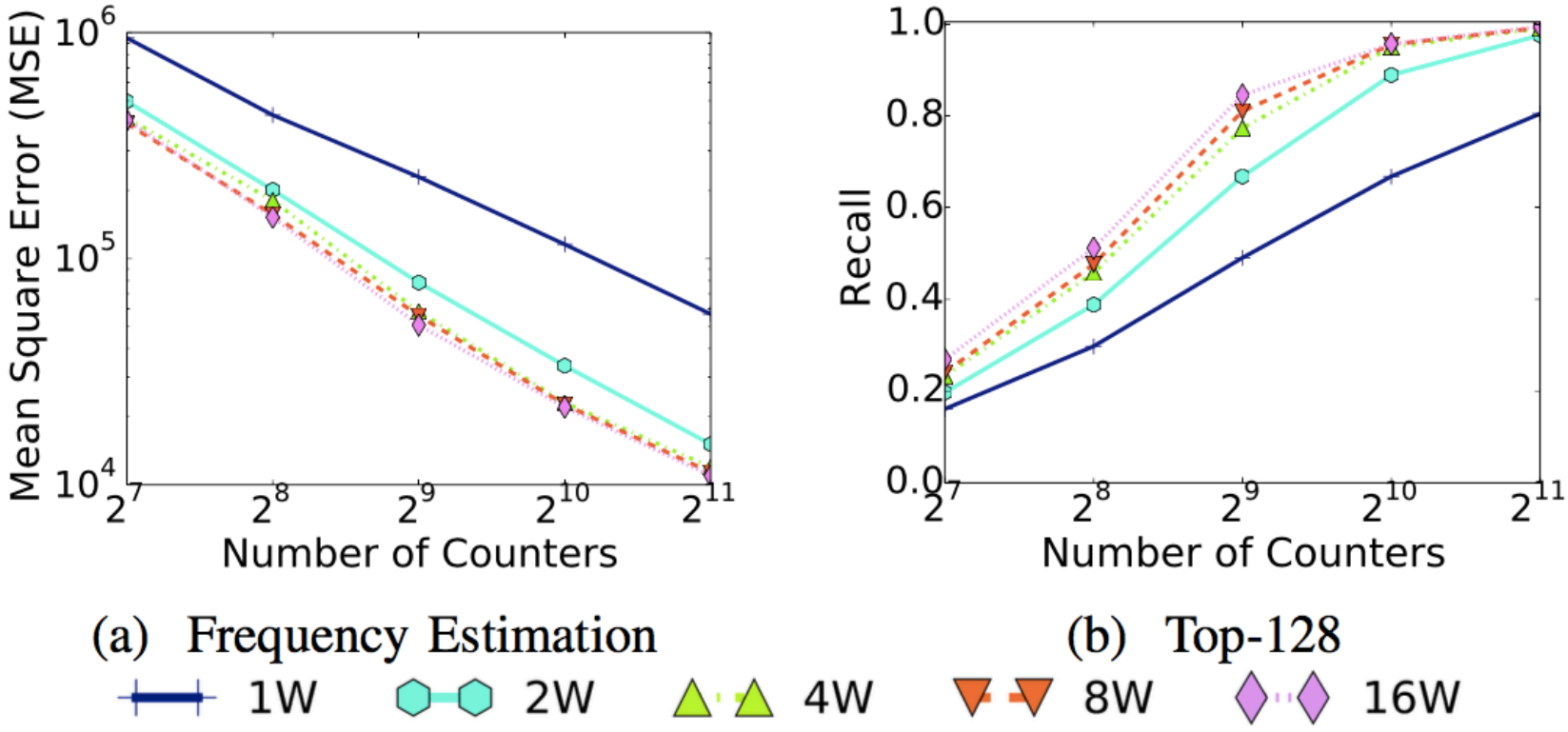
Performance still close to RAP/Space-Saving

Performs better on heavy-tailed workload (network workload)

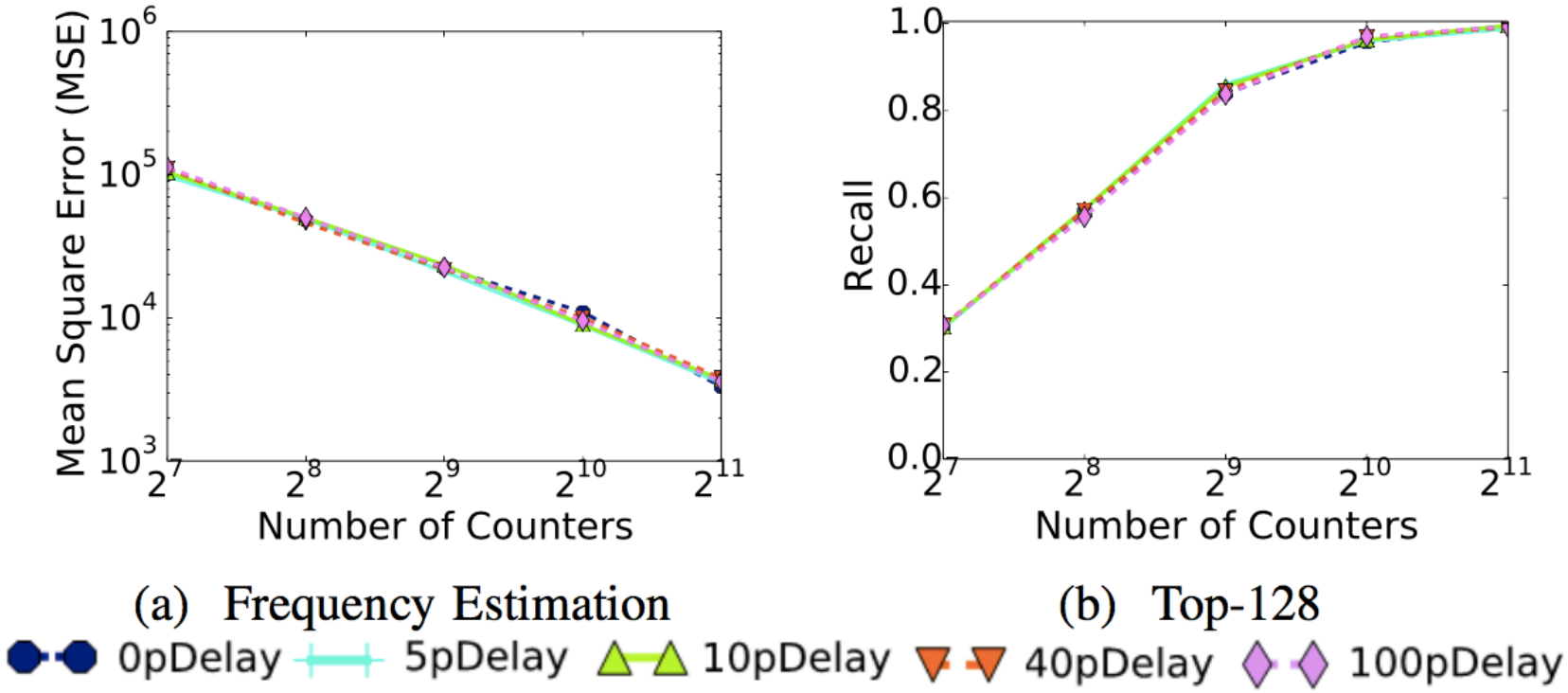
Probabilistic Recirculation



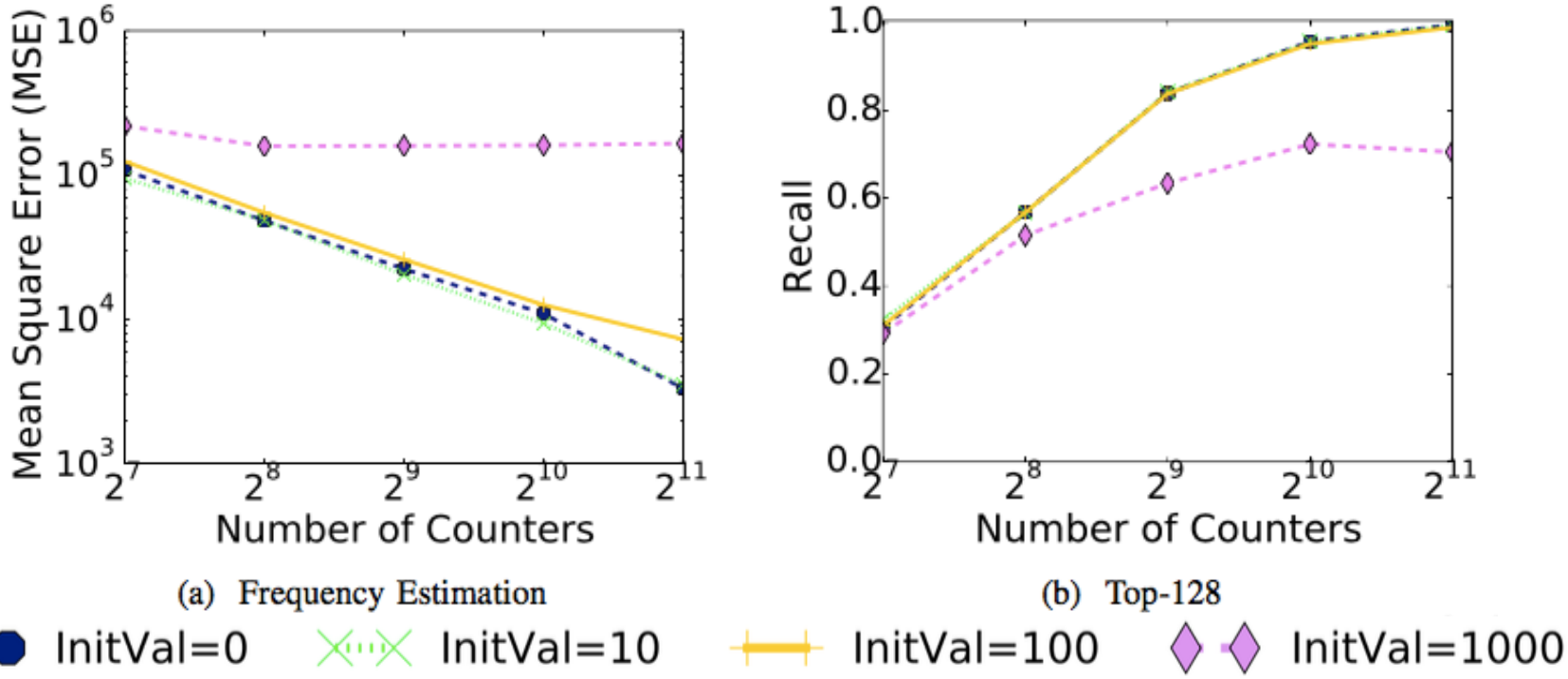
Evaluation 1: how many stages needed?



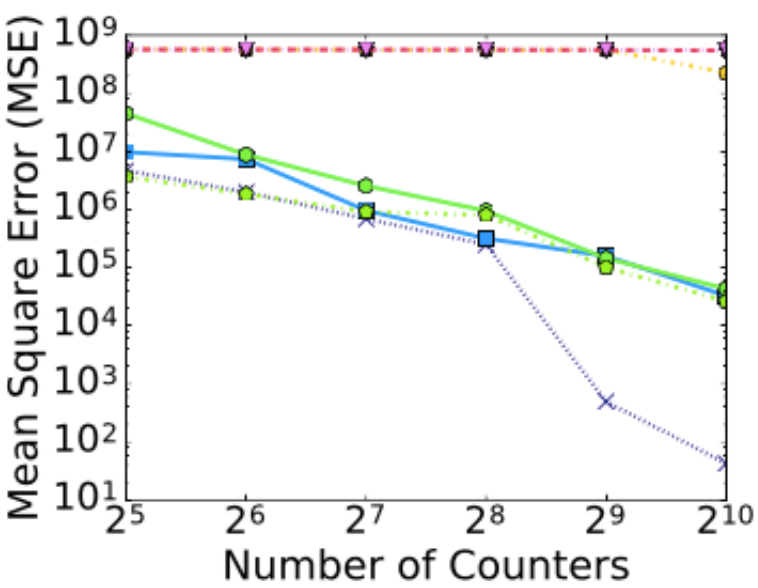
Evaluation 2: recirculation delay?



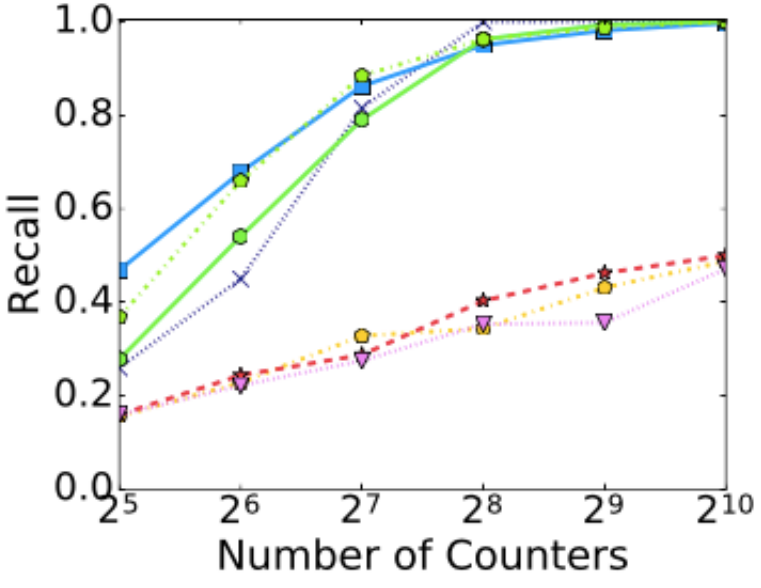
Evaluation 3: initialize counters?



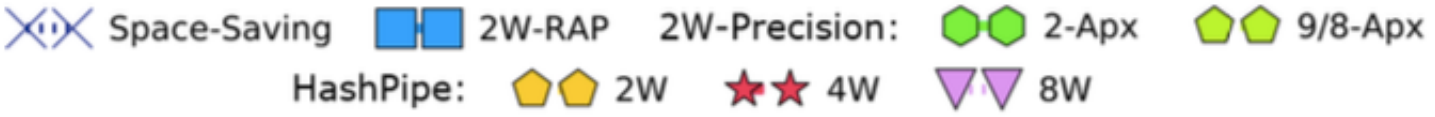
Overall Performance



(b) UCLA



(e) UCLA



Adapt any measurement algorithm to PISA:

- Approximate all input and intermediate variable
- Try to output something better than random
- Expose a resource-accuracy tradeoff

"Any measurement can be pushed into data plane!"
(fine print: answer is approximate.)

Approximation to the rescue

We often care about "important flows", not all flows

Approximate is not indiscriminative sampling!

- Need more memory
- Complex processing, hardware-unfriendly

- Can use sketch, save memory
- Simple logic, hardware-friendly
- Less accurate? (accuracy-memory tradeoff)

Summary of PRECISION

1. Maintains per-flow counters for heavy-hitters

2. Use ***pr****obabilistic* admission to remember new flows

3. Use ***rec****irculation* to simplify register access

4. Almost as-good accuracy, despite PISA HW constraints

Backup slides

Hardware-Friendly == Less Branching?

HashPipe: input
(carried flow ID, carried counter)

- Match carried_ID==register_ID?
- If matched, register_cnt++; else, compare register_cnt<carried_cnt
- If register_cnt is smaller, swap(register_ID, carried_ID) swap(register_cnt, carried_cnt)

Output:
(carried flow ID, carried counter)

Branching within a stage!

Cannot fit in Barefoot Tofino

PRECISION: input
(my flow_id, carried_min)

- Match flow_id==register_ID?
- If matched, register_cnt++; else, compare register_cnt<carried_min
- If register_cnt is smaller, update carried_min, and remember this stage as the "min stage"

Output:
(my flow_id, carried_min)

At the end of pipeline:

- Flip coin based on carried_min, also infer the right stage

When recirculate: easy, just do it…

The stacking trick

Oh, we need 3 hardware stage for a PRECISION "stage"

- Stage A: access register for ID
- Stage B: access register for CNT
- Stage C: remember ST_MIN

In total, 3x hardware stages?

- So many stages to waste!

Can't compress:
B has conditional dependency over A (B>A), also C>B.

Init	1	2	3	4	5	6	7
	1A	1B	1C	2A	2B	2C	3A

Solution: "pipeline" in the pipeline

- Like CPU instruction fetch pipeline

Observation: 2A↛1B, 2B↛1C

"Amortized" 1 hardware stage!
3*x becomes x+3

Init	1	2	3	4	5	6	7
H(1)	1A	1B	1C				
H(2)		2A	2B	2C			
H(3)			3A	3B	3C		
H(4)				4A	4B	4C	
H(5)					5A	5B	5C