A Comparison of Performance and Accuracy of Measurement Algorithms in Software

Omid Alipourfard, Masoud Moshref[1], Yang Zhou[2], Tong Yang[2], Minlan Yu[3]

Yale University, Barefoot Networks[1], Peking University[2], Harvard University[3]

Network function virtualization is trending

Data centers

Use cloud to manage cloud

Edge networks

Mini-clouds at the edge

Firewall **Load balancer** **CDN** **WAN opt.** **NAT**

Network function virtualization is trending

Data centers

Use cloud to manage cloud

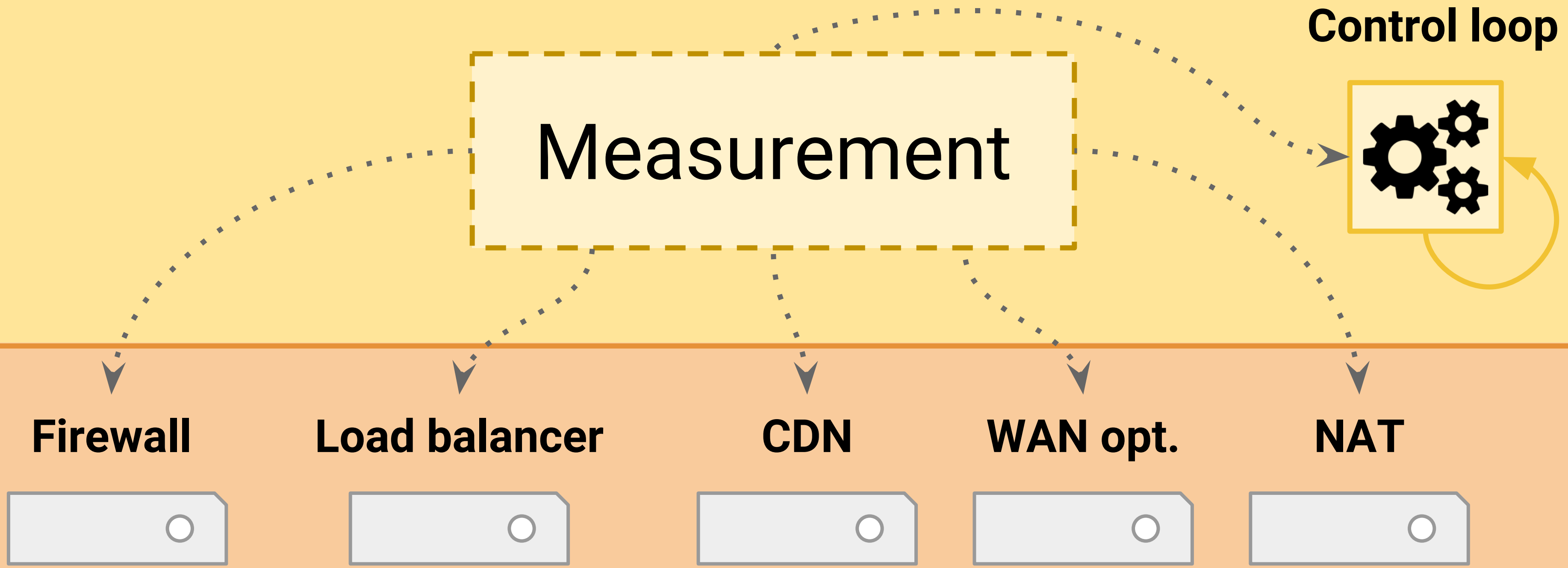
Edge networks

Mini-clouds at the edge

Virtualization, dynamic scale-out, fast iterations ...

Firewall **Load balancer** **CDN** **WAN opt.** **NAT**

Network function virtualization is trending



Measurement algorithms come with many implementations

Measurement Task	Tree/Heap	Sketch	Hash table
Heavy hitter	ANCS ’11, ICDT’ 05	NSDI’ 13, SIGCOMM’ 17	SIGCOMM’ 02
Super spreader		SIGCOMM’ 17, PODS’ 05	IMC’ 10, NDSS’ 05
Flow size distrib.		SIGMETRICS’ 04	IMC’ 10
Change detection	CoNEXT’ 13	TON’ 07	IMC’ 10
Entropy estimation		COLT’ 11	SIGMETRICS’ 06
Quantiles	SIGMOD’ 01, 99, 13	Hot ICE’ 11	

Measurement algorithms come with many implementations

Measurement Task	Tree/Heap	Sketch	Hash table
Heavy hitter	ANCS ’11, ICDT’ 05	NSDI’ 13, SIGCOMM’ 17	SIGCOMM’ 02
Entropy estimation		COLT’ 11	SIGMETRICS’ 06
Quantiles	SIGMOD’ 01, 99, 13	Hot ICE’ 11	

Which algorithm works best for NFs running on software ...

Design concerns for software switches

Domain	Hardware switches	Software switches
Constraint	Limited memory size	
Objective	*Fit in memory*	
Opportunity	Deterministic throughput	

Design concerns for software switches

Domain	Hardware switches	Software switches
Constraint	Limited memory size	Limited cache size
Objective	*Fit in memory*	*Maximize throughput*
Opportunity	Deterministic throughput	Large memory (hierarchical)

Closer look at heavy hitter detection

Find the most popular items (flows) in a packet stream.

Hash table based	Count sketch	Heap based
Update the entry (**e**) in the hash table.	Hash the header ***n*** times and update relevant entries ($\mathbf{e}_s$).	Keep a heap of counters.
		Replace the smallest counter if no space available.
Report if ***e*** > threshold.	Report if min($\mathbf{e}_s$) > threshold.	Report if ***entries*** > threshold.

Closer look at heavy hitter detection

Find the most popular items (flows) in a packet stream.

Hash table based

Update the entry (**e**) in the hash table.

Report if **e** > threshold.

Count sketch

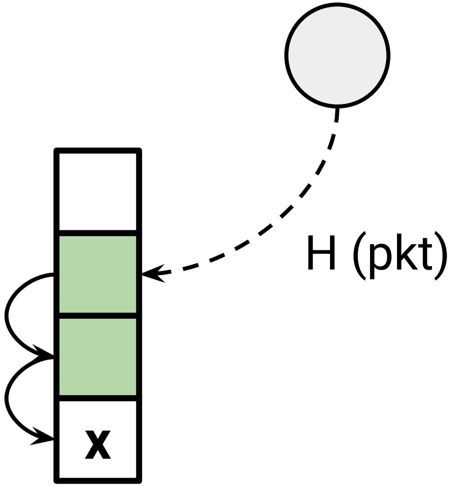
Heap based

Report if ***entries*** > threshold.

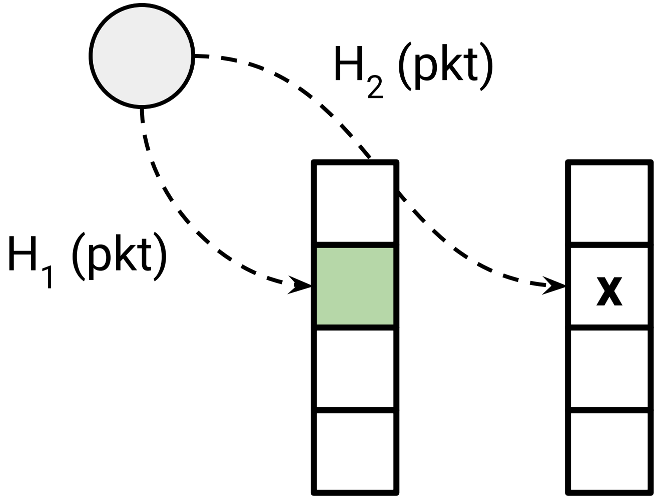
What hash table works best?

Cuckoo vs. linear hash table

Two popular hash tables: *Cuckoo hash table* and *Linear hash table*.



Linear hash table



Cuckoo hash table

Evaluation settings

Settings

- DPDK Framework
- Intel Xeon-E5 2650 v3, 10G NIC
- CAIDA (1.4 mil flows, 40 mil pkts, 64B pkts)
- Zero packet loss test - RFC 2544
- Reporting interval 100ms ~ control loop frequency

Metrics

- Performance: average packet processing time
- We also measure precision/recall in the paper

Linear hashing outperforms Cuckoo hashing

- *Performance:* Linear table is 10~30% faster than Cuckoo table.

Why?

- *Computation:* Two hashes (Cuckoo) vs one hash (Linear).
- *Random access:* Two for Cuckoo vs. one for Linear.

Different from the database world - Memory is not an issue!

- Make the table large so collisions are rare!

Cuckoo vs. linear hash table

Two popular hash tables: *Cuckoo hash table* and *Linear hash table*.

Takeaways

- Use the least # of computations and random memory accesses.
- If you can, use large memory to reduce your computations.

● **Memory is not an issue!** Make the table large so collisions are rare.

Comparison of algorithm classes

Hash table based	Count sketch	Heap based
Update the entry (**e**) in the hash table.	Hash the header ***n*** times and update relevant entries ($\mathbf{e}_s$).	Keep a heap of counters.
Report if **e** > threshold.	Report if min($\mathbf{e}_s$) > threshold.	Replace the smallest counter if no space avail.
		Report if ***entries*** > threshold.

Comparison of algorithm classes

Hash table based	Count sketch	Heap based
Linear hash table	Count sketch with one hash (Count-array)	Heap + Linear hash table

Simplest data structure works best

Hash table based	Count sketch	Heap based
Linear hash table	Count sketch with one hash (Count-array)	Heap + Linear hash table

Results

- Count array is the fastest.
- Hash table performance converges to count-array with larger tables.
- Heap based algorithms are slow because of random memory access.

How general are the results?

- Other measurement tasks
- Other traffic skews
- Amount of data kept per packet/flow
- Shared vs. separate data structure

Results hold for other measurement tasks

Change detection ⟷ **Superspreader detection**

Computationally heavy — Memory heavy

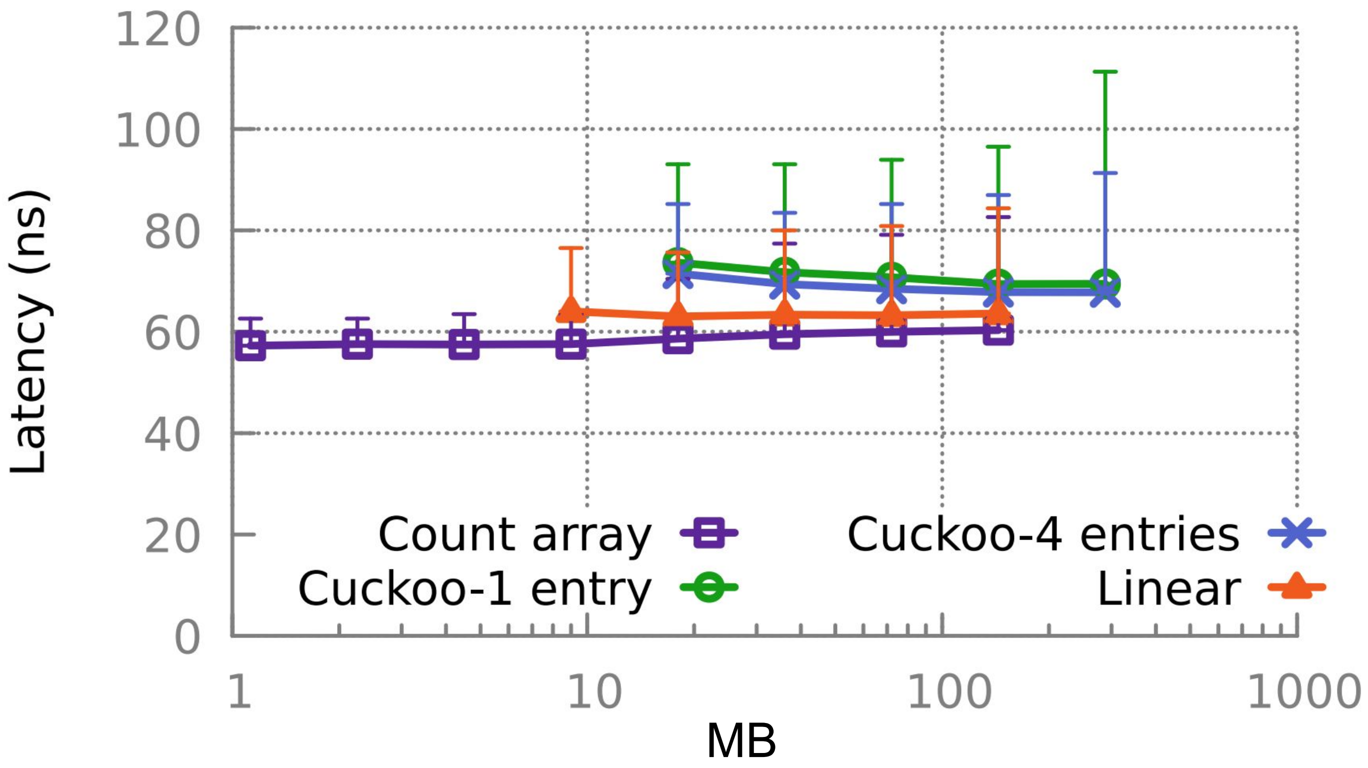
Model flow's traffic

Report flows outside model's predictions

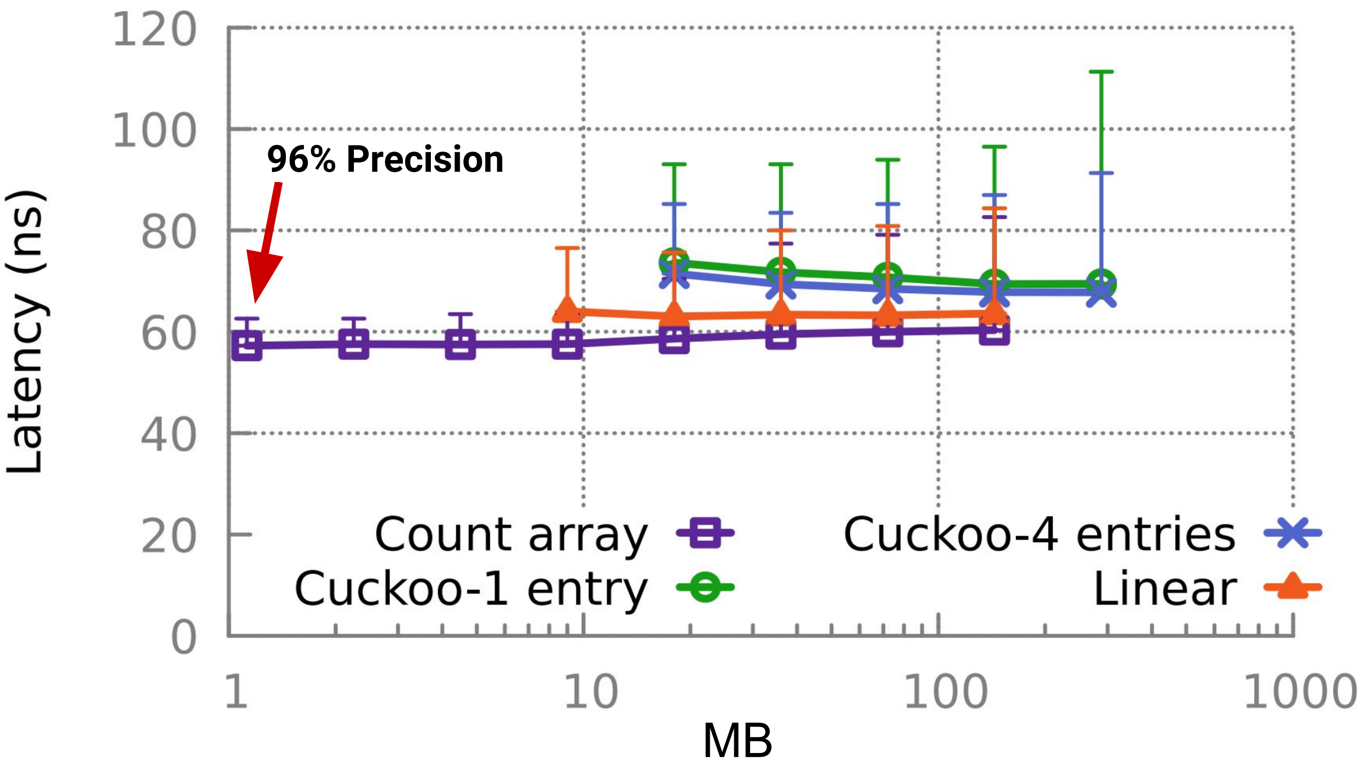
Update a bloom filter per packet

Does CPU behave differently dealing with other measurement task types?

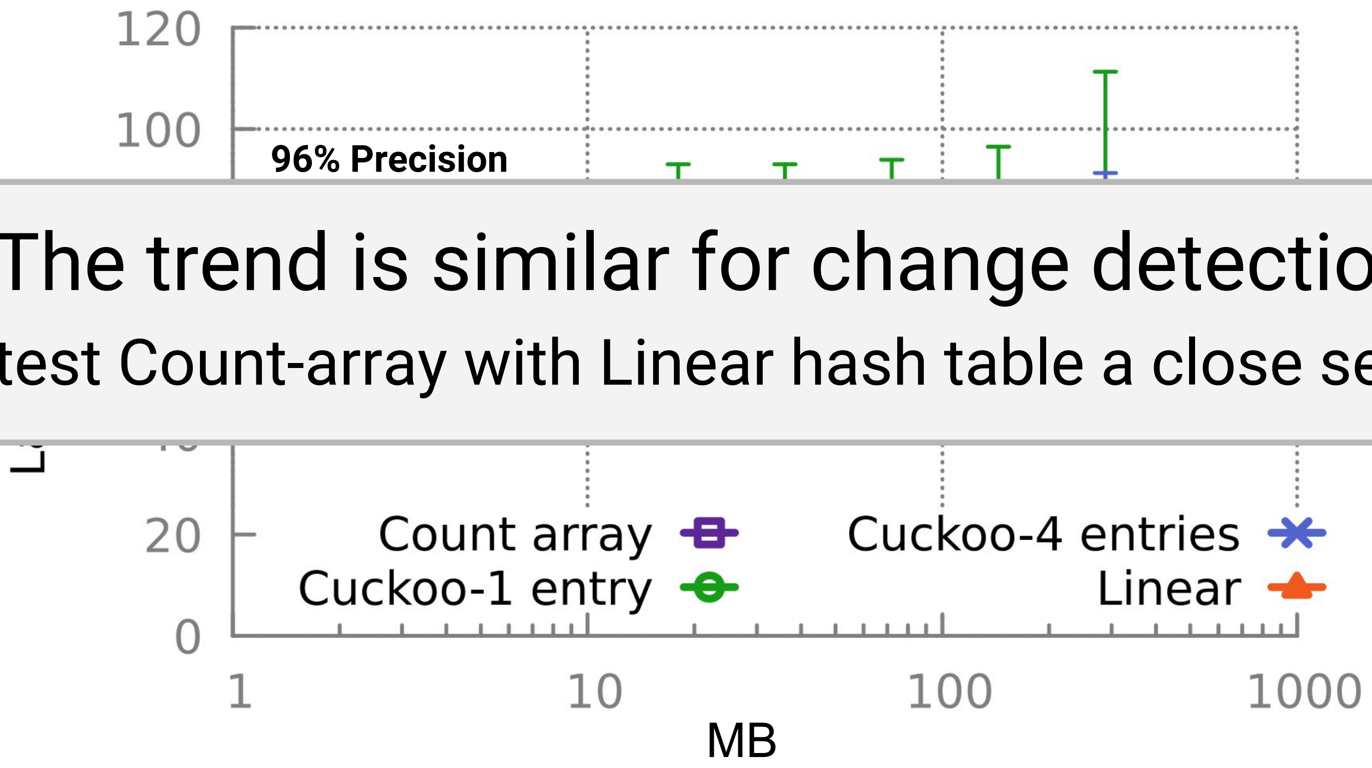
Superspreaders: Count-array is the fastest



Superspreaders: Count-array is the fastest



Superspreaders: Count-array is the fastest



The trend is similar for change detection:

Fastest Count-array with Linear hash table a close second.

Impact of traffic skew on latency

Concerns

- Working set gets larger with lower skew.
- More items read in cache per packet batch.

Impact of traffic skew on latency

Concerns

- Working set gets larger with lower skew.
- More items read in cache per packet batch.

Observations

- Perf. degradation depends on the # of memory accesses per pkt.
- Count-array and linear hash table still the fastest.

Impact of bytes kept per flow on latency

Concerns

- Less number of items fit in the cache.
- Traverse multiple cache lines on a miss.

Impact of bytes kept per flow on latency

Concerns

- Less number of items fit in the cache.
- Traverse multiple cache lines on a miss.

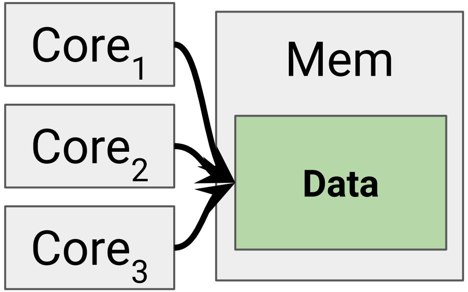
Observations

- 1.9x higher latency - 4 bytes (70ns~) to 60 bytes (130ns~)
- **Solution:** Separate keys and values in the hash table.
  - 1.16x higher latency - 4 byte (90ns~) to 60 byte (105ns~)

Impact of shared/separate data-structure

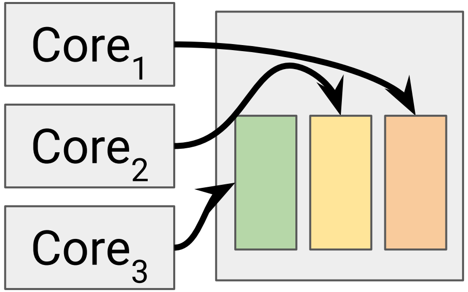
Shared: Easy to report measured results.

- More cache bouncing between cores.



Separate: Merging to report is difficult.

- No cache bouncing between cores.



Impact of shared/separate data-structure

Observations

Sharing is expensive.

- Cache bouncing causes L3 latency for most memory accesses.
- Does not scale to many cores.

Merging is cheap.

- Very low memory bandwidth (even at 10ms reporting intervals).

Conclusions

Measurement in software servers is different than hardware:

- Use more memory to do less computation.
- Reduce data pulled into the cache per packet.

Calls for new:

- **Algorithms**, e.g., “sketch” over computation not memory.
- **Data structures**, e.g., seq. access pattern to match the CPU arch.

Thanks!

The code and benchmarks are available at:

https://github.com/SiGe/measure-pkt

LUX ET VERITAS

PEKING UNIVERSITY

1898

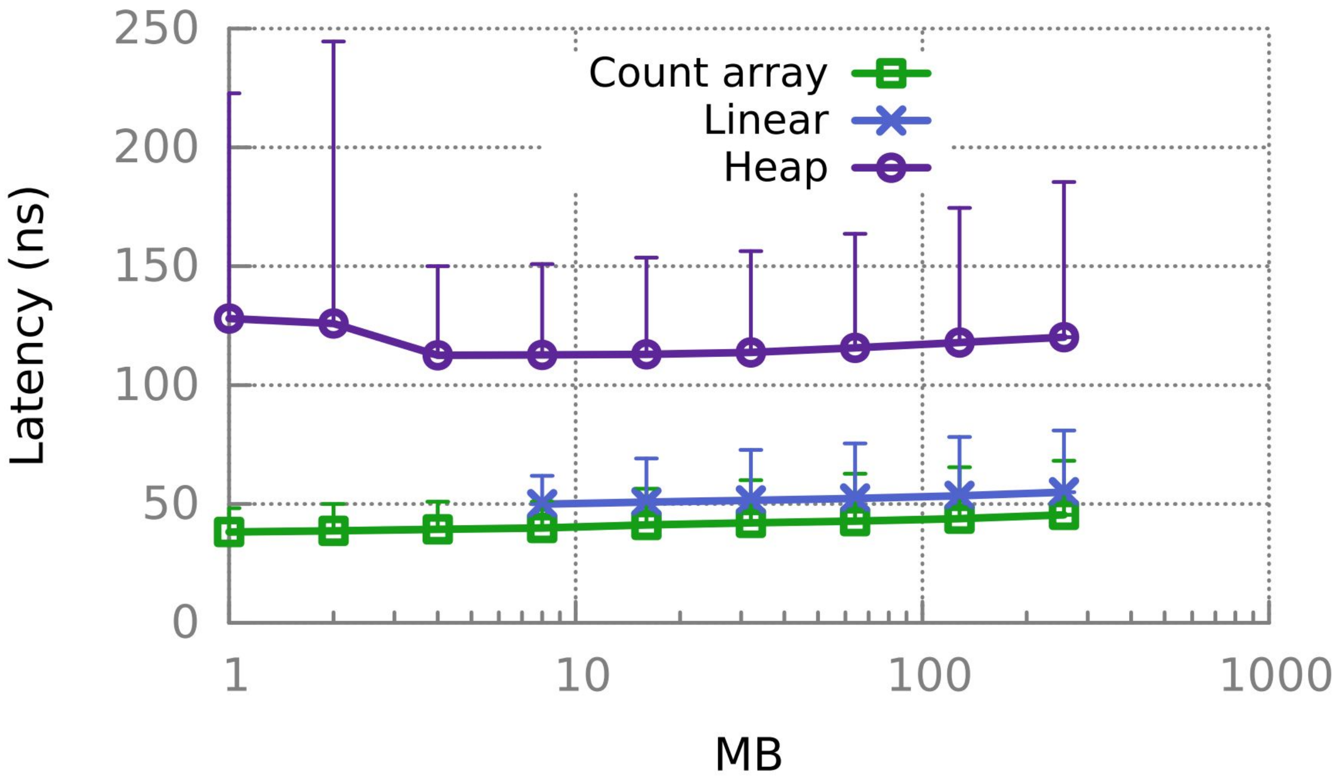
Simplest data structure works best

Hash table based	Count sketch	Heap based
Linear hash table	Count sketch with one hash	Heap + Linear hash table

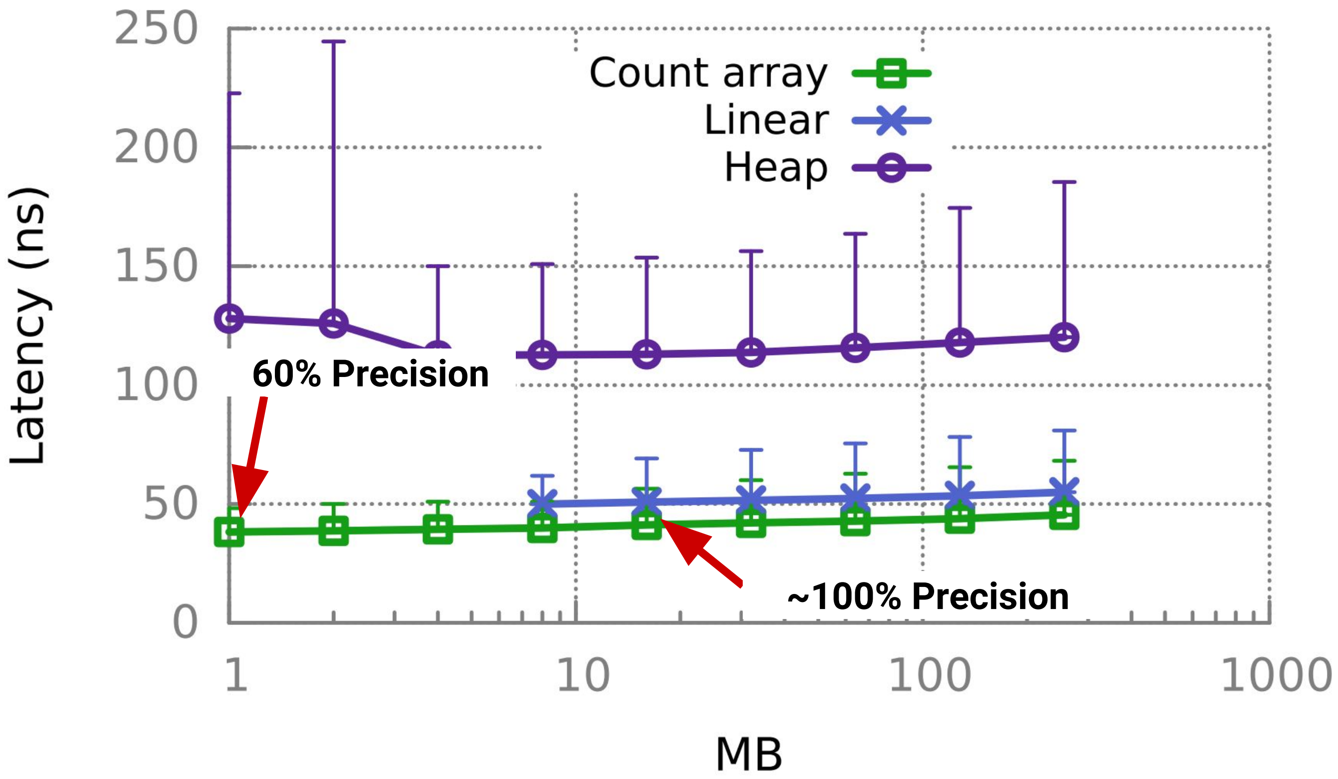
Least amount of computation wins.

- Count array is the fastest.
- Hash table performance converges to count-array with larger tables.
- Heap based algorithms are slow because of random memory access.

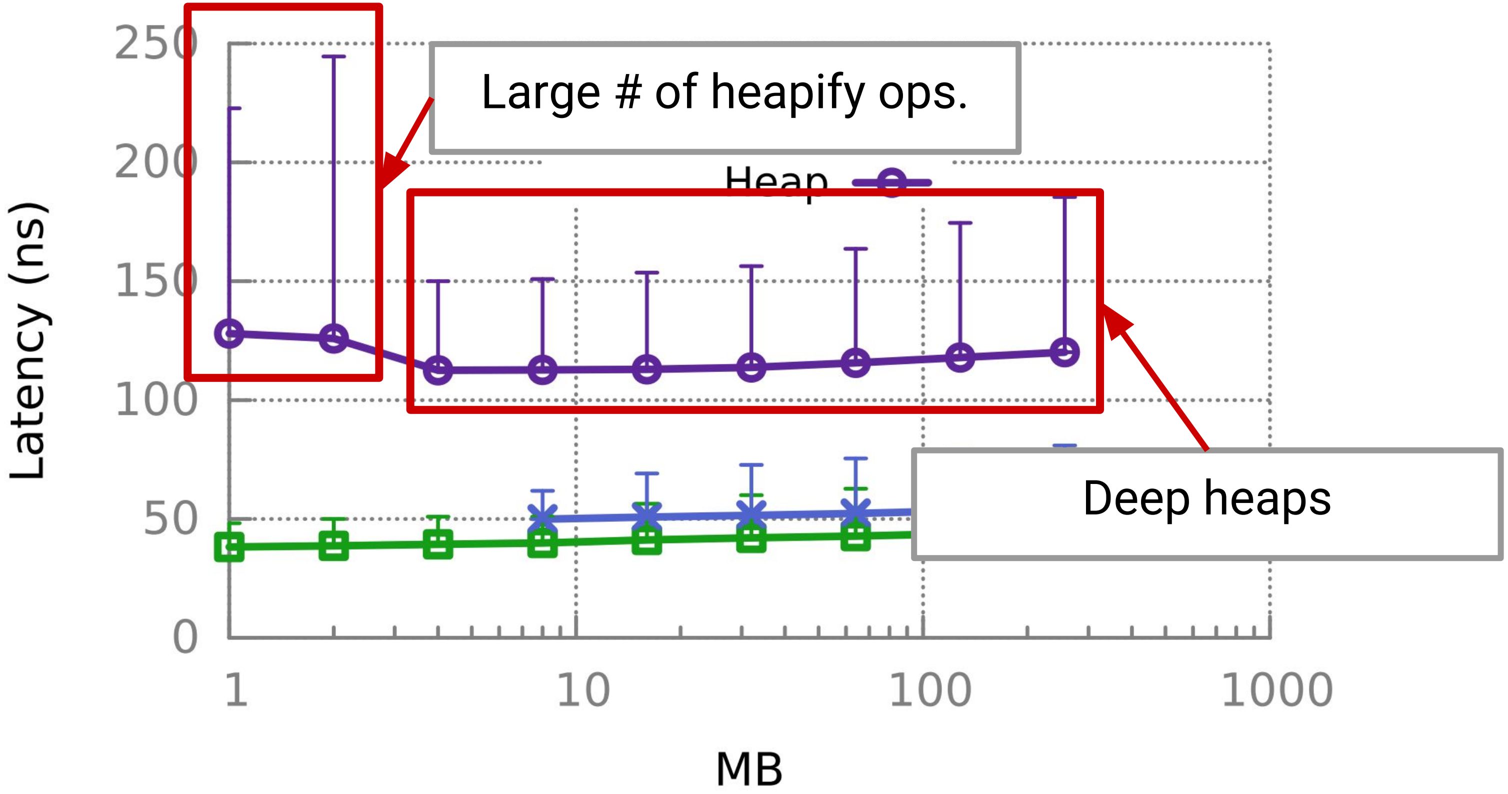
Change-detection: Count-array is the fastest



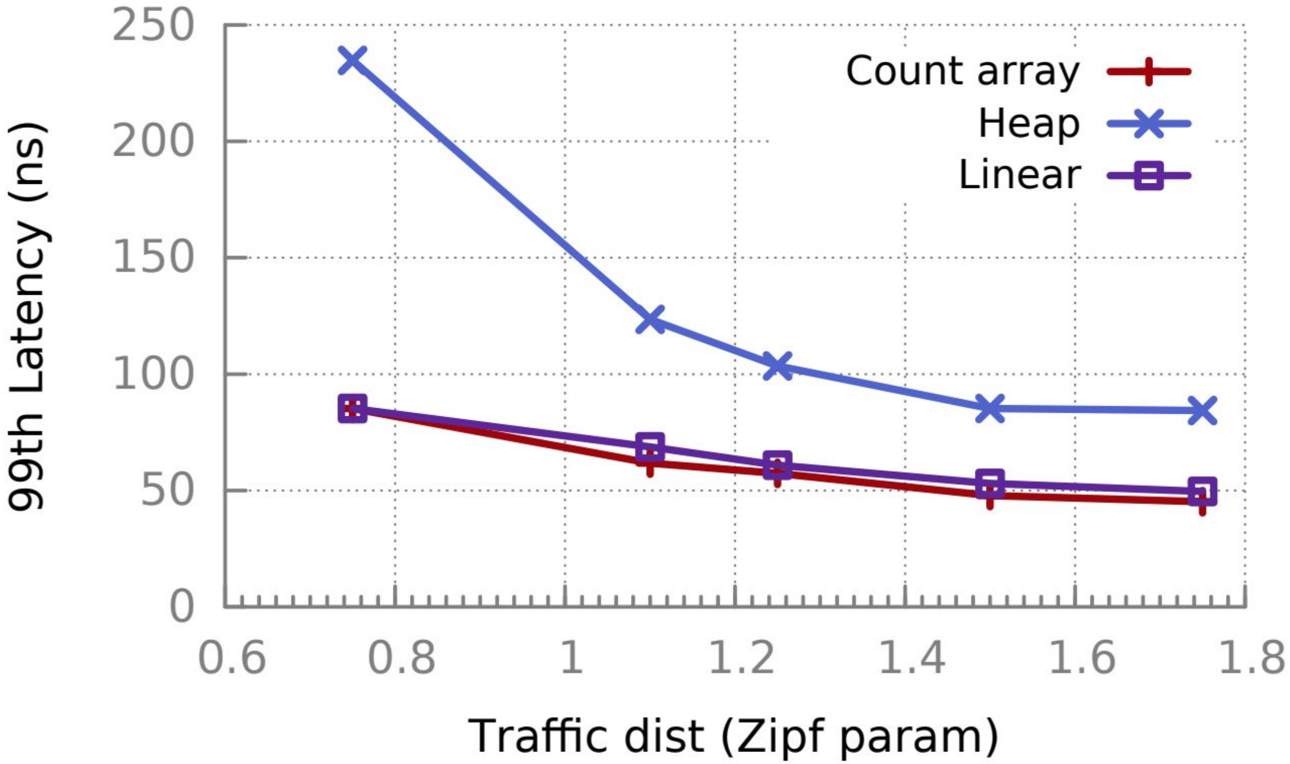
Change-detection: Count-array is the fastest



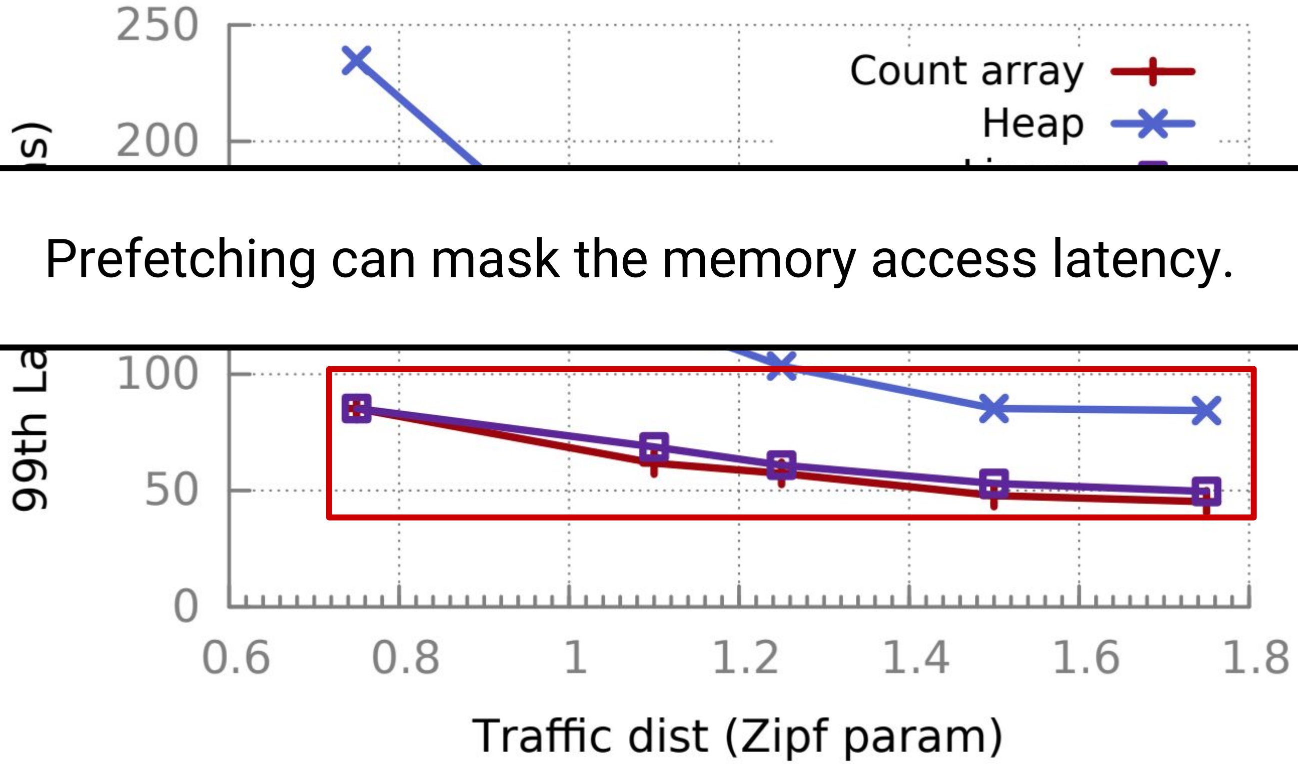
Change-detection: Count-array is the fastest



Impact of traffic skew on latency

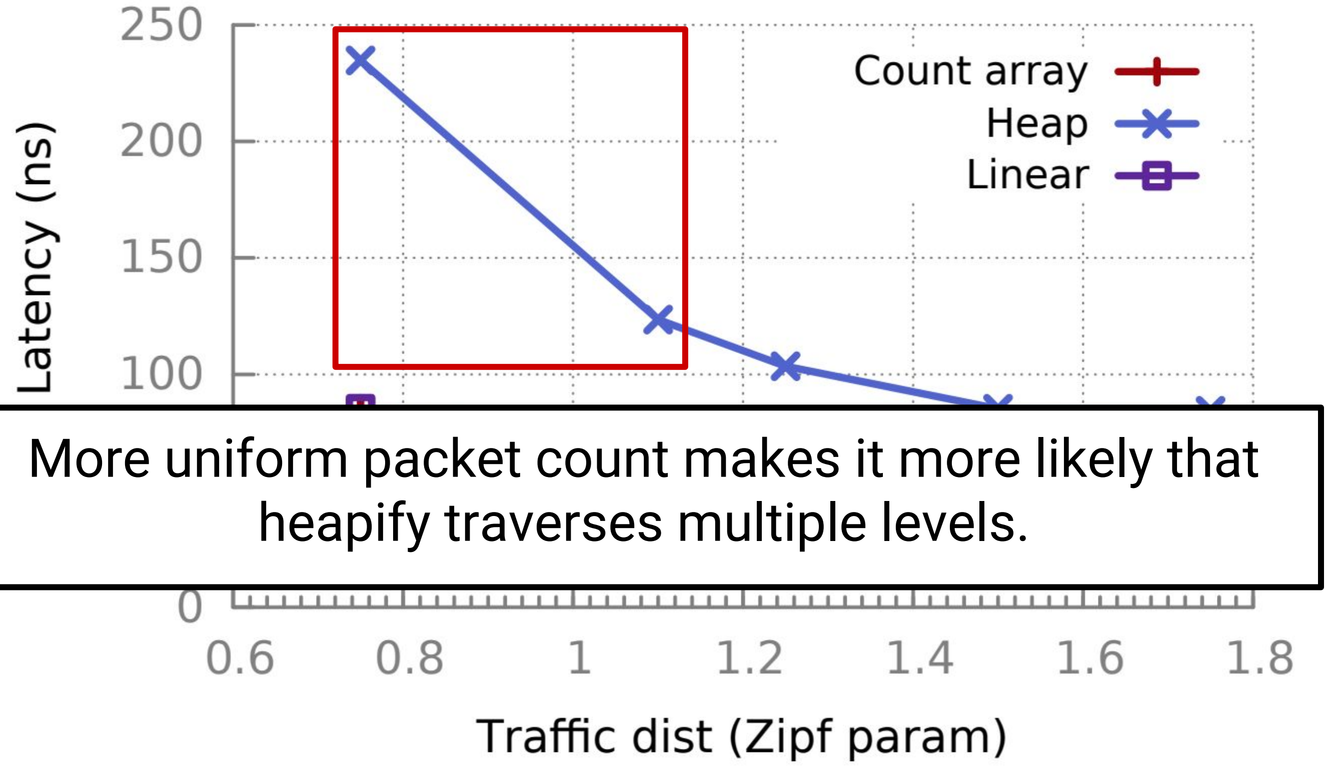


Impact of traffic skew on latency



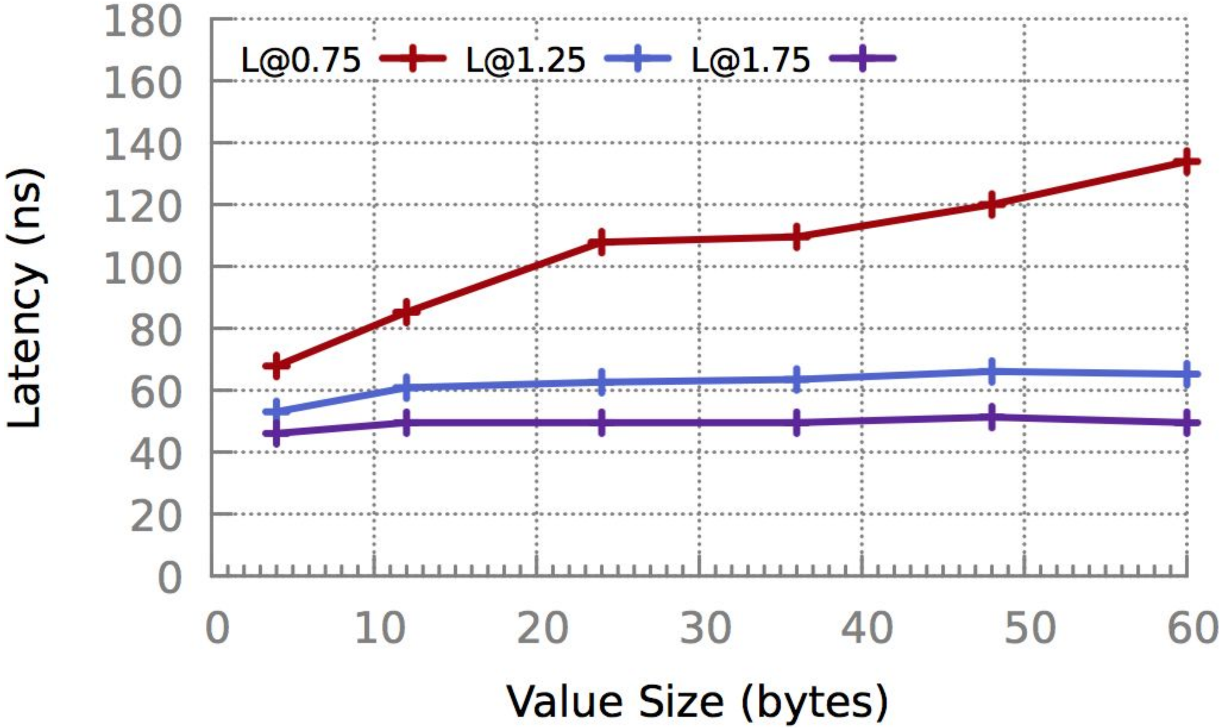
Prefetching can mask the memory access latency.

Impact of traffic skew on latency

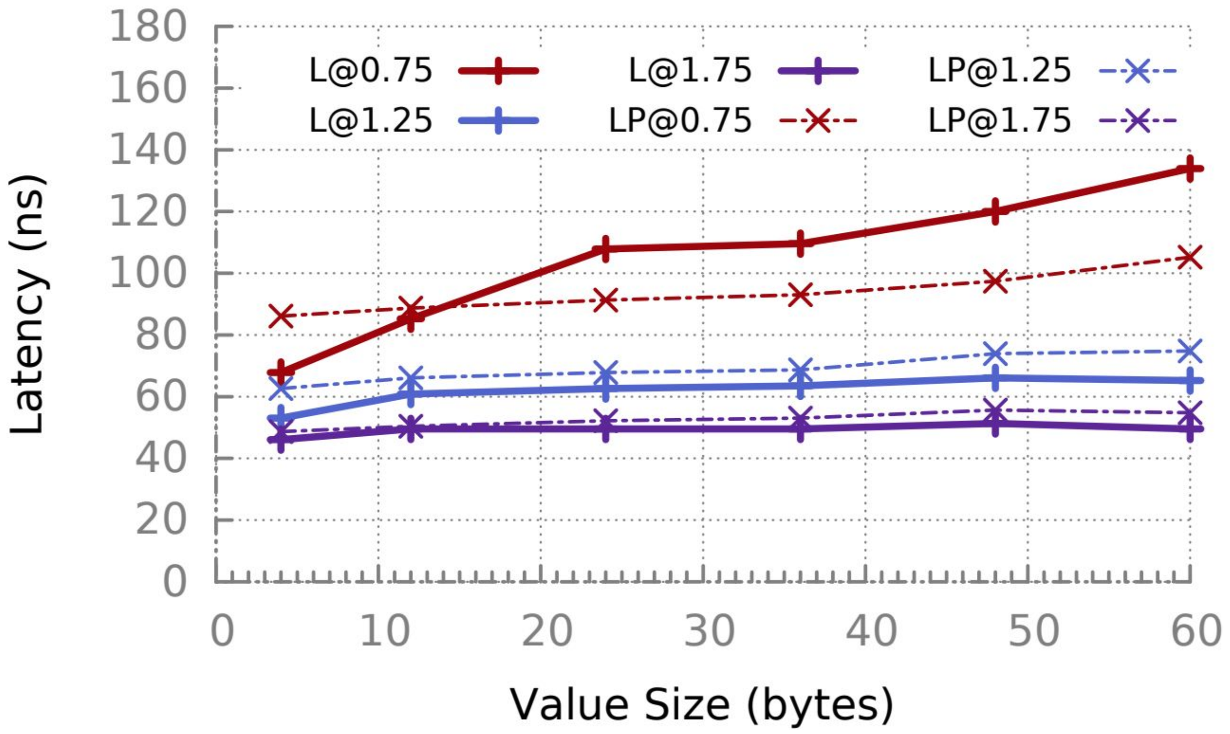


More uniform packet count makes it more likely that heapify traverses multiple levels.

Bytes fetched impacts the performance



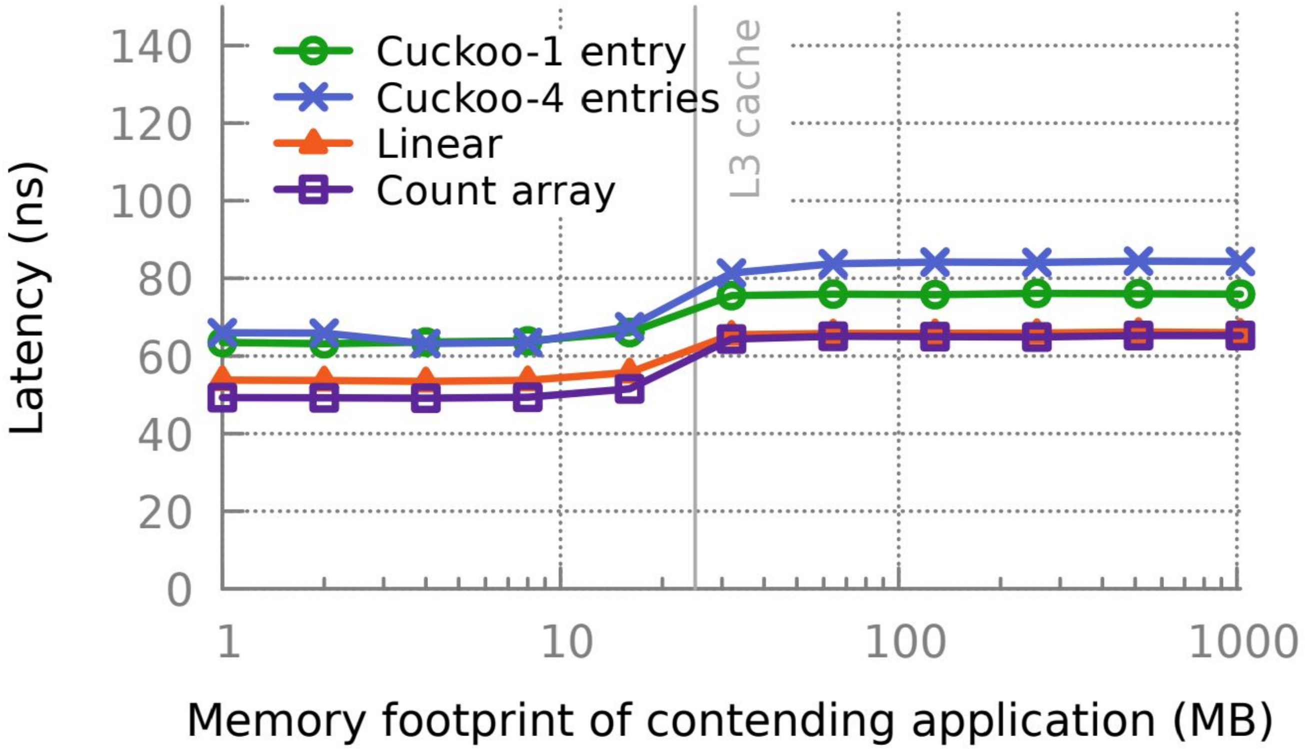
Mask the latency by keeping the values away



Impact of other apps. on measurement

- **Cache exhaustion**: working set not fitting in memory.
- **Memory BW exhaustion:** higher latency to fetch data.

Impact of other apps. on measurement



Impact of other apps. on measurement