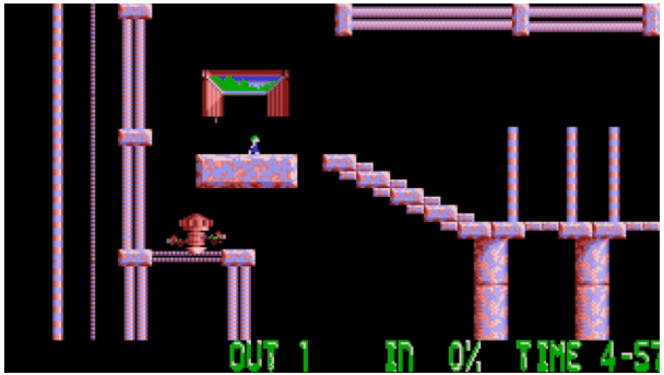
Fast Reroute in P4:
Keep Calm and Enjoy Programmability



Marco Chiesa
KTH Royal Institute of Technology

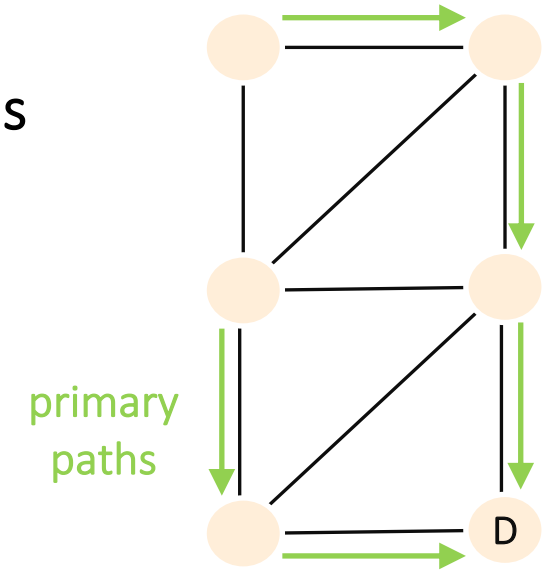
Joint project with:

Roshan Sedar
Michael Borokhovich
Gianni Antichi
Stefan Schmid

Fast Reroute (FRR)

What is FRR?

- forwarding rules **conditional** on port status
- e.g., IP Loop-Free Alternates



Fast Reroute (FRR)

What is FRR?

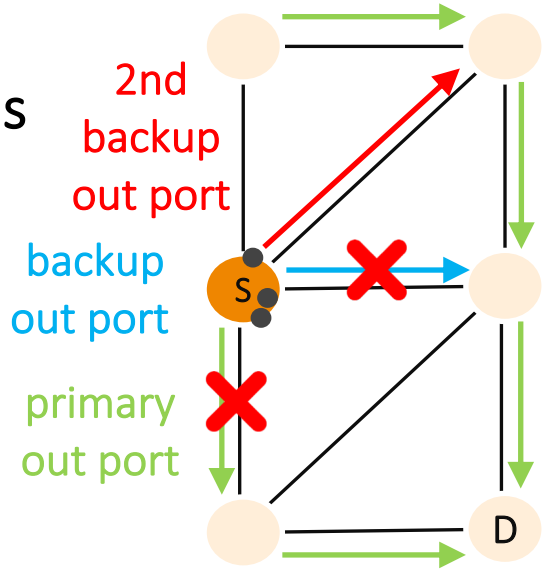
- forwarding rules **conditional** on port status
- e.g., IP Loop-Free Alternates

Pros:

- **reduce** network downtime (≤50ms)
  - no need to invoke control-plane

Cons:

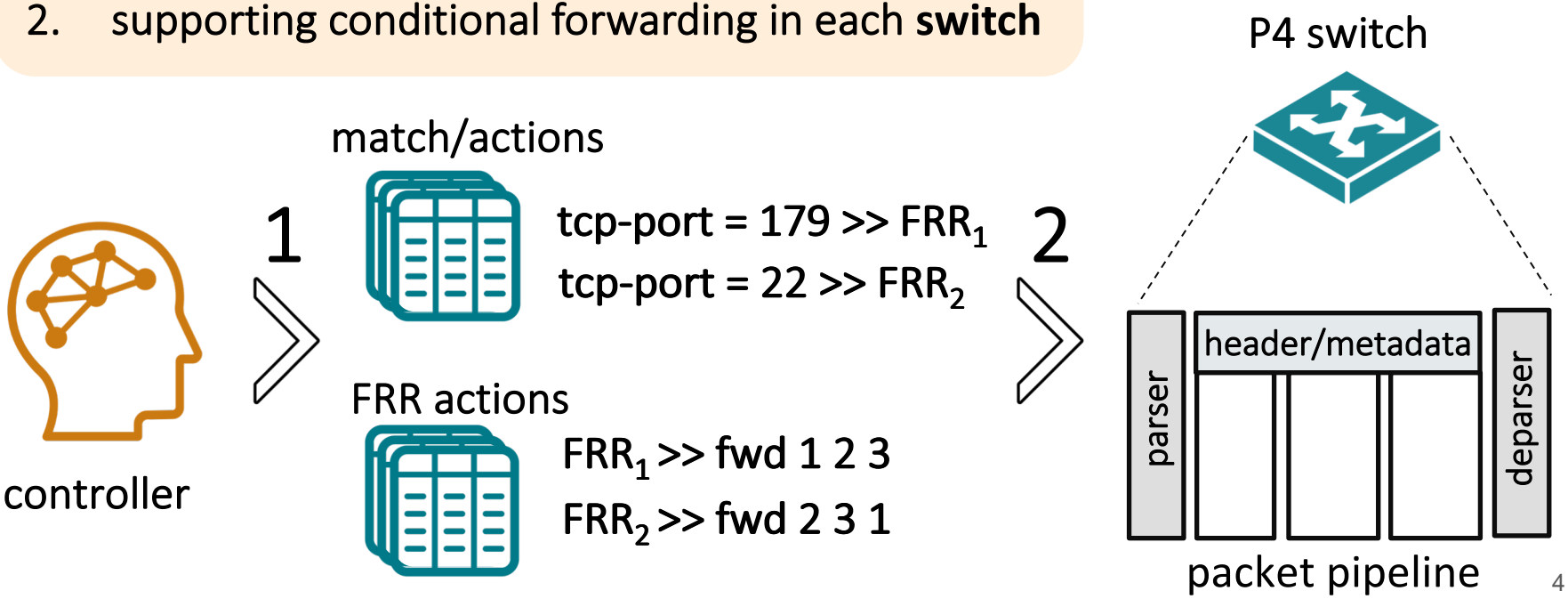
- **increase** forwarding space occupancy
  - proactively stores backup forwarding rules



How do we implement FRR in P4?

FRR entails solving two orthogonal problems:

1. computing **network-wide** conditional rules
2. supporting conditional forwarding in each **switch**



Supporting conditional forwarding in P4 switch

How do we implement FRR in P4?

FRR entails solving two orthogonal problems:

1. computing **network-wide** conditional rules
2. supporting conditional forwarding in each **switch**

Output

controller

1

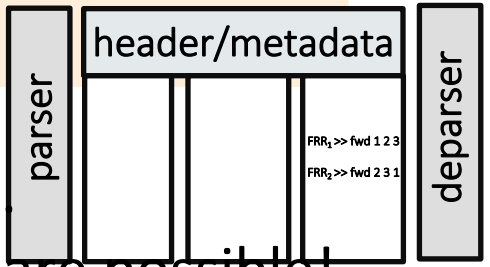
match/actions

tcp-port = 179 >> FRR_1

tcp-port = 22 >> FRR_2

2

P4 switch



packet pipeline

FRR actions

FRR_1 >> fwd 1 2 3

FRR_2 >> fwd 2 3 1

No P4 built-in FRR primitive but…

a **programmable** pipeline: many approaches are possible!

1st approach: recirculation

Input

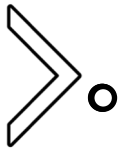
FRR_1 = 1 2 3 4

FRR_2 = 2 3 4 1

FRR_3 = 3 4 1 2

FRR_4 = 4 1 2 3

a set of circular FRR actions



match	action	
out	fwd →	write & recirculate ↻
1	1	out := 2
2	✗	out := 3
3	✗	out := 4
4	4	out := 1

port 2 fails

port 3 fails

throughput reduction

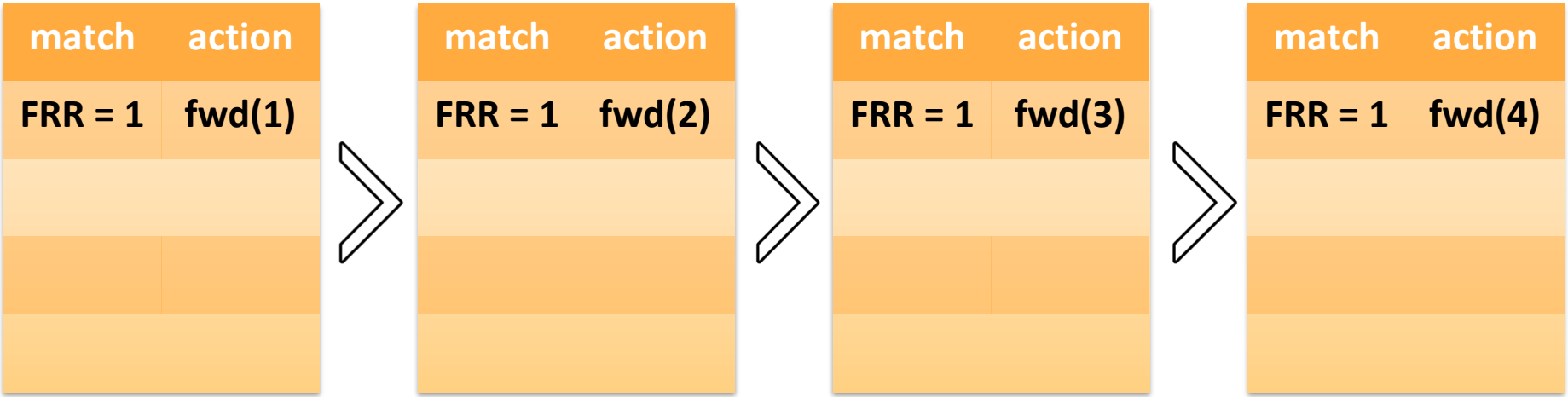
1st requirement for FRR primitive

high
throughput

Second approach: sequential search

Input

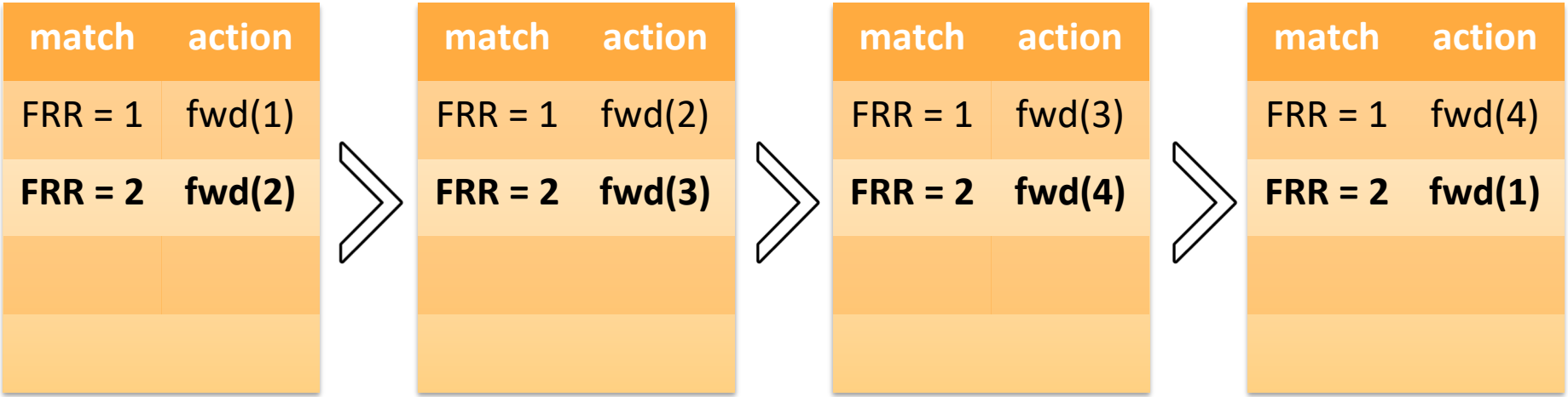
FRR_1 = 1 2 3 4 FRR_2 = 2 3 4 1 FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3



Second approach: sequential search

Input

FRR_1 = 1 2 3 4 **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3



Second approach: sequential search

Input

FRR_1 = 1 2 3 4 FRR_2 = 2 3 4 1 **FRR_3 = 3 4 1 2** FRR_4 = 4 1 2 3

match	action
FRR = 1	fwd(1)
FRR = 2	fwd(2)
FRR = 3	**fwd(3)**

match	action
FRR = 1	fwd(2)
FRR = 2	fwd(3)
FRR = 3	**fwd(4)**

match	action
FRR = 1	fwd(3)
FRR = 2	fwd(4)
FRR = 3	**fwd(1)**

match	action
FRR = 1	fwd(4)
FRR = 2	fwd(1)
FRR = 3	**fwd(2)**

Second approach: sequential search

Input

FRR_1 = 1 2 3 4 FRR_2 = 2 3 4 1 FRR_3 = 3 4 1 2 **FRR_4 = 4 1 2 3**

match	action
FRR = 1	fwd(1)
FRR = 2	fwd(2)
FRR = 3	fwd(3)
FRR = 4	**fwd(4)**

match	action
FRR = 1	fwd(2)
FRR = 2	fwd(3)
FRR = 3	fwd(4)
FRR = 4	**fwd(1)**

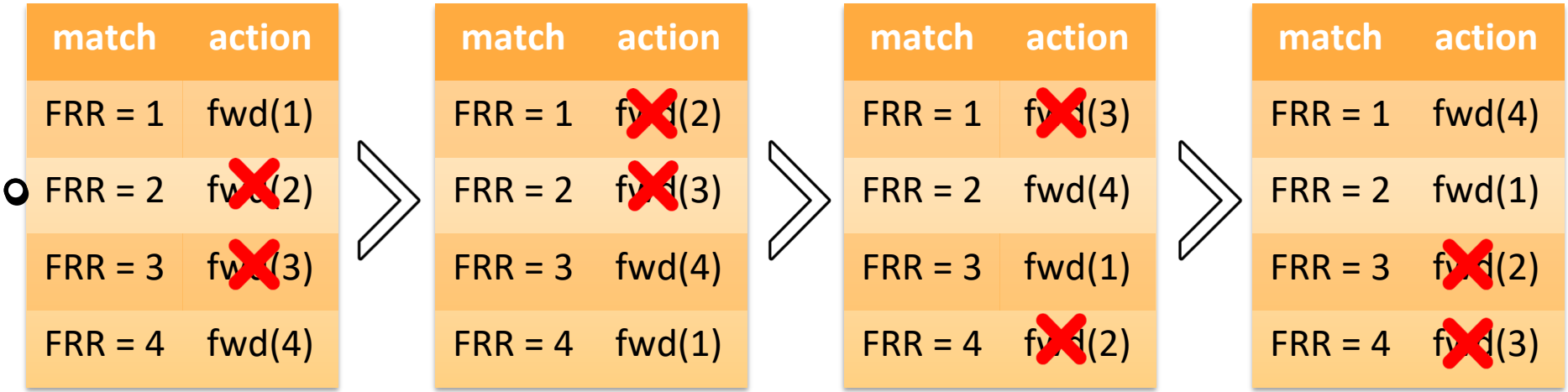
match	action
FRR = 1	fwd(3)
FRR = 2	fwd(4)
FRR = 3	fwd(1)
FRR = 4	**fwd(2)**

match	action
FRR = 1	fwd(4)
FRR = 2	fwd(1)
FRR = 3	fwd(2)
FRR = 4	**fwd(3)**

Second approach: sequential search

Input

$FRR_1 = 1\ 2\ 3\ 4$ $FRR_2 = 2\ 3\ 4\ 1$ $FRR_3 = 3\ 4\ 1\ 2$ $FRR_4 = 4\ 1\ 2\ 3$



port 2 fails

port 3 fails

increased latency
waste of resources at each stage

2nd requirements for FRR primitive

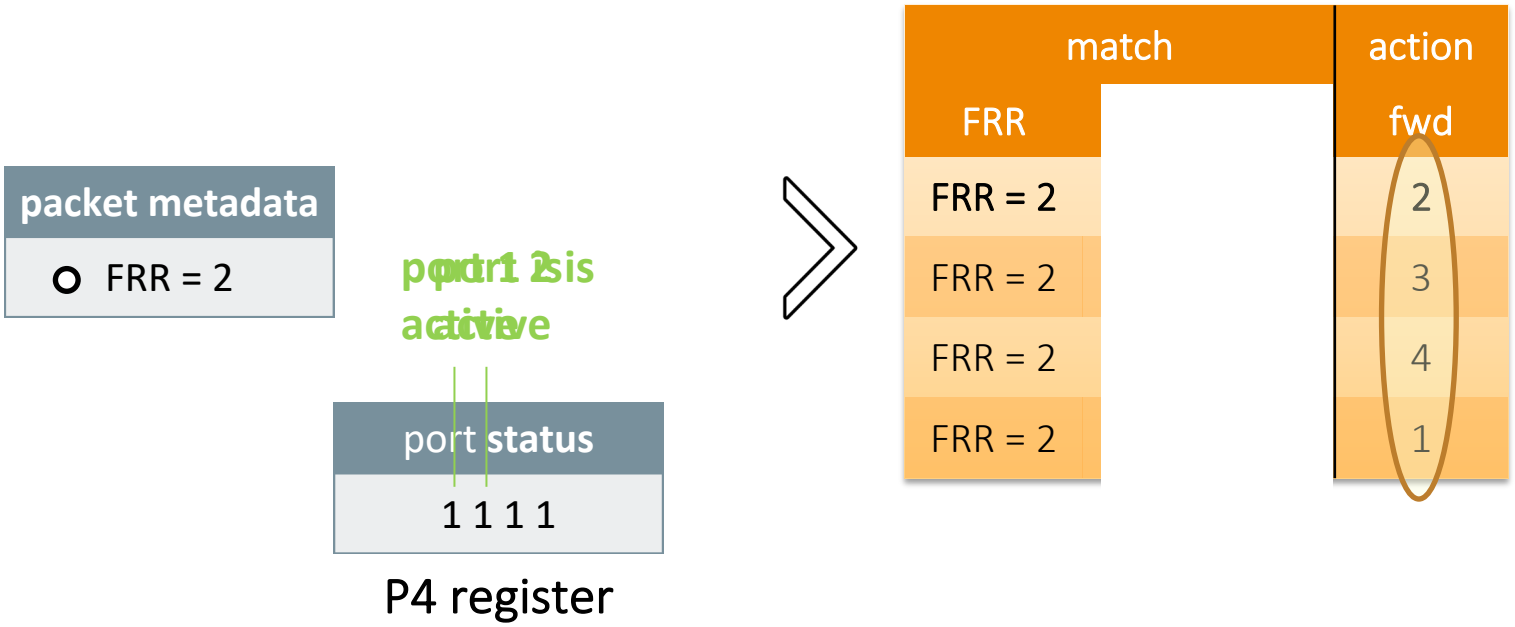
high
throughput

low forwarding
latency

Parallelism needed! Naïve TCAM approach

Input

$FRR_1 = 1\ 2\ 3\ 4$ **$FRR_2 = 2\ 3\ 4\ 1$** $FRR_3 = 3\ 4\ 1\ 2$ $FRR_4 = 4\ 1\ 2\ 3$



Parallelism needed! Naïve TCAM approach

Input

FRR_1 = 1 2 3 4 **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

port 2 fails

packet metadata	port status
FRR = 2	1 0 1 1

port status
1 0 1 1

P4 register

match		action
FRR	port status	fwd
FRR = 2	* 1 * *	2
FRR = 2	*** * 1 ***	**3**
FRR = 2	* * * 1	4
FRR = 2	1 * * *	1

Parallelism needed! Naïve TCAM approach

Input

FRR_1 = 1 2 3 4 **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

ports 2 and 3 fail

packet metadata	port status
FRR = 2	1 0 0 1

port status
1 0 0 1

P4 register

match		action
FRR	port status	fwd
FRR = 2	* 1 * *	2
FRR = 2	* * 1 *	3
FRR = 2	*** * * 1**	**4**
FRR = 2	1 * * *	1

Parallelism needed! Naïve TCAM approach

Input

$FRR_1 = 1\ 2\ 3\ 4$ $FRR_2 = 2\ 3\ 4\ 1$ $FRR_3 = 3\ 4\ 1\ 2$ $FRR_4 = 4\ 1\ 2\ 3$

packet metadata	port status
FRR = 4	1 0 0 1

port status
1 0 0 1

P4 register

match		action
FRR	port status	fwd
FRR = 1	1 * * *	1
FRR = 1	* 1 * *	2
FRR = 1	* * 1 *	3
FRR = 1	* * * 1	4
...	...	...
FRR = 4	*** * * 1**	**4**
FRR = 4	1 * * *	1
FRR = 4	* 1 * *	2
FRR = 4	* * 1 *	3

Final requirements for FRR primitive

high throughput

low forwarding latency

efficient reroute

”port status” P4 register

How much state for “naïve TCAM”?

Input:

- switch with *k* ports
- 10 circular set of FRR actions

Naïve TCAM FRR:

- k^2 number of TCAM entries

For *k* = 24

- 5.760 TCAM entries!

For *k* = 48

- 23.040 TCAM entries!

- 10 pods in a datacenter with F10 FRR [nsdi-13]
- 10 destinations with the ”k arc-disjoint” FRR mechanism [ton-16]

[nsdi-13] *V. Liu et al. "F10: A Fault-Tolerant Engineered Network" in NSDI 2013*
[ton-16] *M. Chiesa et al. "On the Resiliency of Randomized Routing Against Multiple Edge Failures" in Transactions on Networking 2016*

Encoding FRR in the packet metadata

Input

$FRR_1 = 1\ 2\ 3\ 4$ **$FRR_2 = 2\ 3\ 4\ 1$** $FRR_3 = 3\ 4\ 1\ 2$ $FRR_4 = 4\ 1\ 2\ 3$

packet metadata	port status
FRR = 2	1 1 1 1

frr_ports
$b_1 b_2 b_3 b_4 b_5 b_6 b_7$

Encoding FRR input:

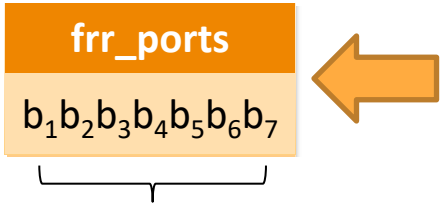
- add a packet metadata field *frr_ports*
- map bits to the switch ports

Encoding FRR in the packet metadata

Input

FRR_1 = 1 2 3 4 **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

packet metadata	port status
FRR = 2	1 1 1 1



Encoding FRR input:

- add a packet metadata field *frr_ports*
- map bits to the switch ports
- set bit to 1 to include a port
- set bit to 0 to skip a port

Encoding FRR in the packet metadata

Input

$FRR_1 = 1\ 2\ 3\ 4$ $FRR_2 = 2\ 3\ 4\ 1$ $FRR_3 = 3\ 4\ 1\ 2$ $FRR_4 = 4\ 1\ 2\ 3$

packet metadata	port status
FRR = 2	1 1 1 1

match	action write frr_ports
FRR = 1	1 1 1 1 0 0 0
FRR = 2	0 1 1 1 1 0 0
FRR = 3	0 0 1 1 1 1 0
FRR = 4	0 0 0 1 1 1 1

bit-to-port mapping is: 1 2 3 4 1 2 3

Encoding FRR input:

- add a packet metadata field *frr_ports*
- map bits to the switch ports
- set bit to 1 to include a port
- set bit to 0 to skip a port

Encoding FRR in the packet metadata

Input

$FRR_1 = 1\ 2\ 3\ 4$ $FRR_2 = 2\ 3\ 4\ 1$ $FRR_3 = 3\ 4\ 1\ 2$ $FRR_4 = 4\ 1\ 2\ 3$

packet metadata	port status
FRR = 2	1 1 1 1

match	action
frr_ports	fwd
1 * * * * * *	1

match	action write frr_ports
FRR = 1	1 1 1 1 0 0 0
FRR = 2	0 1 1 1 1 0 0
FRR = 3	0 0 1 1 1 1 0
FRR = 4	0 0 0 1 1 1 1

bit-to-port mapping is: 1 2 3 4 1 2 3

Encoding FRR in the packet metadata

Input

FRR_1 = 1 2 3 4 **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

packet metadata	port status
FRR = 2	1 1 1 1

match	action write frr_ports
FRR = 1	1 1 1 1 0 0 0
FRR = 2	**0 1 1 1 1 0 0**
FRR = 3	0 0 1 1 1 1 0
FRR = 4	0 0 0 1 1 1 1

bit-to-port mapping is: 1 2 3 4 1 2 3

match		action
frr_ports	port status	fwd
1 * * * * * *	1 * * *	1
* **1** * * * * *	* **1** * *	**2**
* * 1 * * * *	* * 1 *	3
* * * 1 * * *	* * * 1	4
* * * * 1 * *	1 * * *	1
* * * * * 1 *	* 1 * *	2
* * * * * * 1	* * 1 *	3

Encoding FRR in the packet metadata

Input

FRR_1 = 1 2 3 4 **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

packet metadata	port status
FRR = 2	1 **0** 1 1

port 2 fails

match	action write frr_ports
FRR = 1	1 1 1 1 0 0 0
FRR = 2	**0 1 1 1 1 0 0**
FRR = 3	0 0 1 1 1 1 0
FRR = 4	0 0 0 1 1 1 1

match		action
frr_ports	port status	fwd
1 * * * * * *	1 * * *	1
* 1 * * * * *	* 1 * *	2
* * **1** * * * *	* * **1** *	**3**
* * * 1 * * *	* * * 1	4
* * * * 1 * *	1 * * *	1
* * * * * 1 *	* 1 * *	2
* * * * * * 1	* * 1 *	3

bit-to-port mapping is: 1 2 3 4 1 2 3

Encoding FRR in the packet metadata

Input

FRR_1 = 1 2 3 4 **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

packet metadata	port status
FRR = 2	1 **0 0** 1

ports 2 and 3 fail

match	action write frr_ports
FRR = 1	1 1 1 1 0 0 0
FRR = 2	**0 1 1 1 1 0 0**
FRR = 3	0 0 1 1 1 1 0
FRR = 4	0 0 0 1 1 1 1

match		action
frr_ports	port status	fwd
1 * * * * * *	1 * * *	1
* 1 * * * * *	* 1 * *	2
* * 1 * * * *	* * 1 *	3
* * * **1** * * *	* * * **1**	**4**
* * * * 1 * *	1 * * *	1
* * * * * 1 *	* 1 * *	2
* * * * * * 1	* * 1 *	3

bit-to-port mapping is: 1 2 3 4 1 2 3

Encoding FRR in the packet metadata

Input

FRR_1 = 1 2 3 4 FRR_2 = 2 3 4 1 FRR_3 = 3 4 1 2 **FRR_4 = 4 1 2 3**

packet metadata	port status
FRR = 4	1 1 1 1

match	action write frr_ports
FRR = 1	1 1 1 1 0 0 0
FRR = 2	0 1 1 1 1 0 0
FRR = 3	0 0 1 1 1 1 0
FRR = 4	**0 0 0 1 1 1 1**

bit-to-port mapping is: 1 2 3 4 1 2 3

match		action
frr_ports	port status	fwd
1 * * * * * *	1 * * *	1
* 1 * * * * *	* 1 * *	2
* * 1 * * * *	* * 1 *	3
* * * **1** * * *	* * * **1**	**4**
* * * * 1 * *	1 * * *	1
* * * * * 1 *	* 1 * *	2
* * * * * * 1	* * 1 *	3

How much space does this encoding save?

Input:

- switch with *k* ports
- 10 circular set of FRR actions

Without encoding:

- k^2 number of TCAM entries

With encoding:

- *2k-1* number of TCAM entries

For *k* = 24

- **92%** less TCAM entries
- 470 instead of 5.760

For *k* = 48

- **96%** less TCAM entries
- 950 instead of 23.040

Implementing existing FRR mechanisms

Why circular FRR sequences?

- Provide resiliency to multiple link failures with small overhead

F10 FRR [NSDI'13]:

- iterates through upward and downward datacenter links

Depth-First-Search FRR [HotSDN'13]:

- iterates through children nodes

K-arc disjoint FRR [Infocom'16]:

- iterates through *k* spanning trees

[nsdi-13] *V. Liu et al. "F10: A Fault-Tolerant Engineered Network" in NSDI 2013*
[HotSDN'13] *Borokhovich et al ""Graph exploration algorithms" in HotSDN 2013*
[ton-16] *M. Chiesa et al. "On the Resiliency of Randomized Routing Against Multiple Edge Failures" in Transactions on Networking 2016*

Implementing existing FRR mechanisms

Why circular FRR sequences?

- Provide resiliency to multiple link failures with small overhead

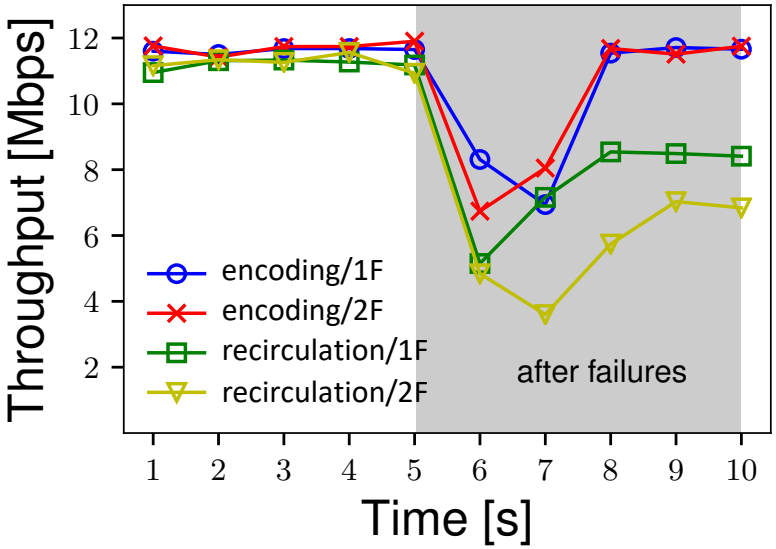
Depth-First-Search FRR [HotSDN'13]:

- iterates through children nodes
- caution: skip parent node

OpenFlow vs P4 (k = 48):

- 56.448 vs 190 TCAM entries
- **99.7%** reduction

Preliminary Mininet evaluation:



Towards a P4 FRR primitive

high throughput

low forwarding latency

efficient reroute

small forwarding state

Transform FRR input into a P4 program:

- based on a mix of naïve and encoding TCAM-based approaches
- many challenging optimization problems

Summary

Fast Reroute is a critical functionality in today's network

- requires high throughput, low latency, fast reactivity, small state overhead

P4 does not define an FRR built-in primitive

- compilers must program the P4 pipeline

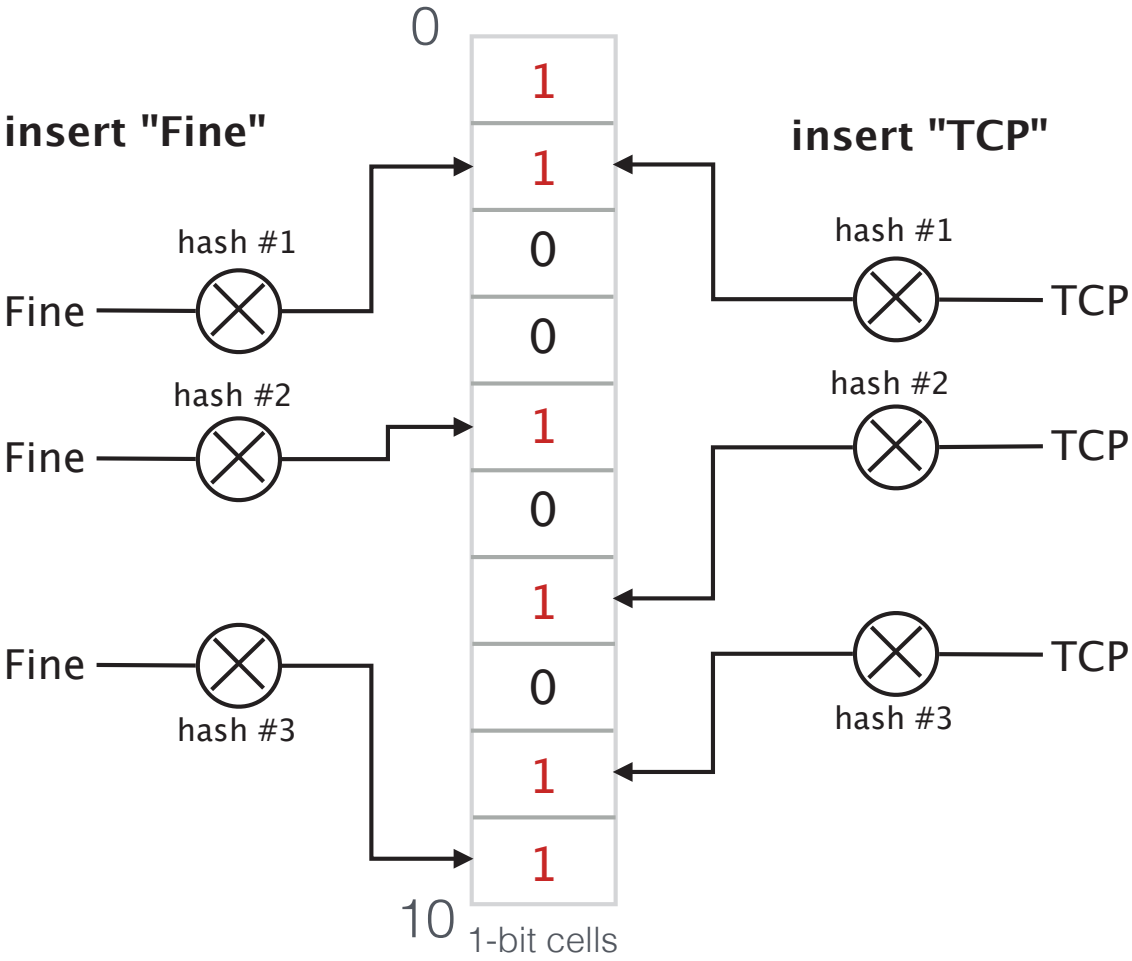
We propose a relatively simple TCAM-based FRR primitive

- based on bit-level mapping of ports
- no FRR-tailored hardware support
- future work:
  - optimize mapping computation
  - evaluation on real switches

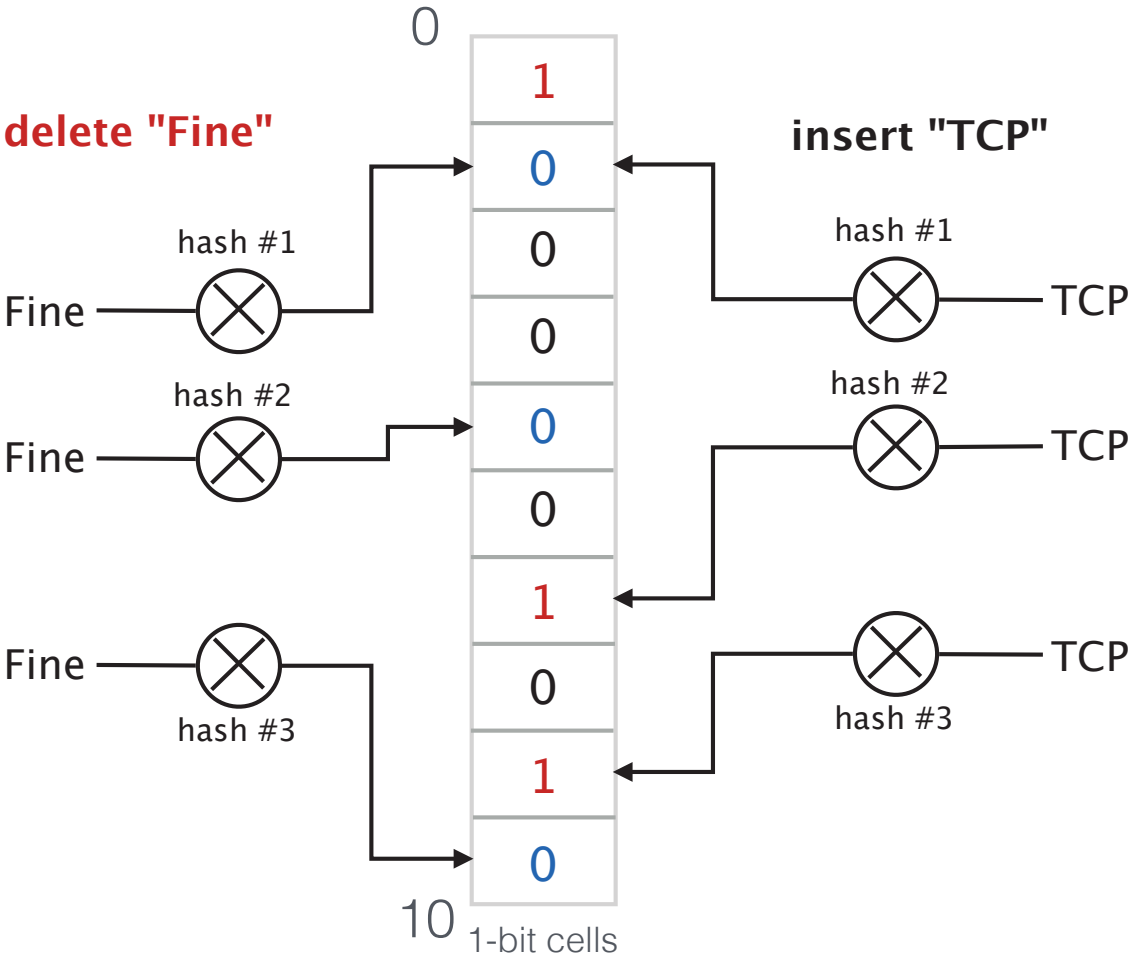
R. Sedar et al.
*"**Supporting Emerging Applications With Low-Latency Failover in P4**"*
*In ACM SIGCOMM workshop **NEAT 2018***

Thank you!

However Bloom Filters do not handle deletions



However Bloom Filters do not handle deletions



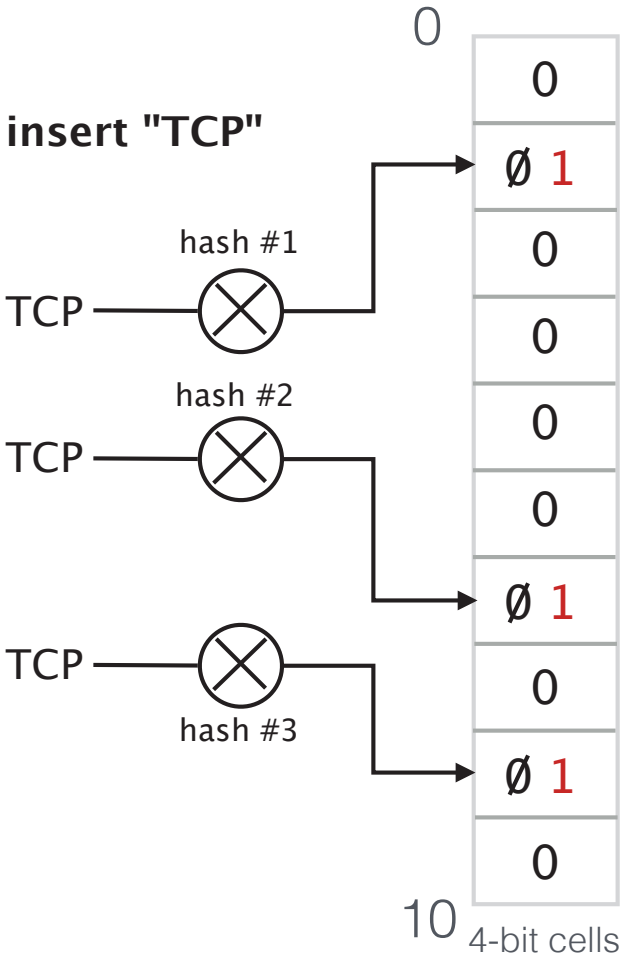
If deleting an element means resetting 1s to 0s, then deleting "Fine" also deletes "TCP"

But we can easily extend them to handle deletions

This extended version is called a Counting Bloom Filter

But we can easily extend them to handle deletions

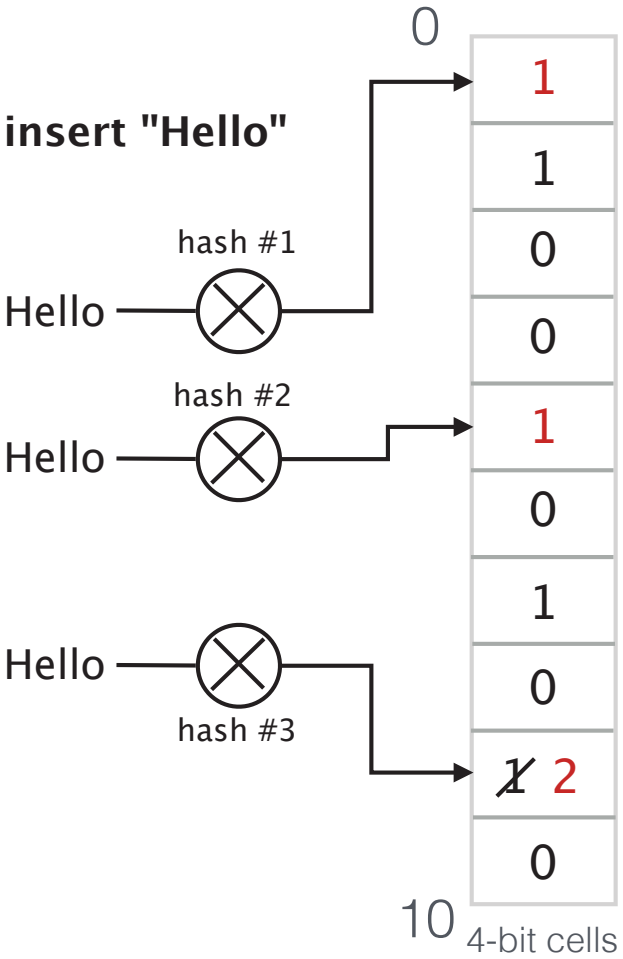
This extended version is called a Counting Bloom Filter



To add an element, increment the corresponding counters

But we can easily extend them to handle deletions

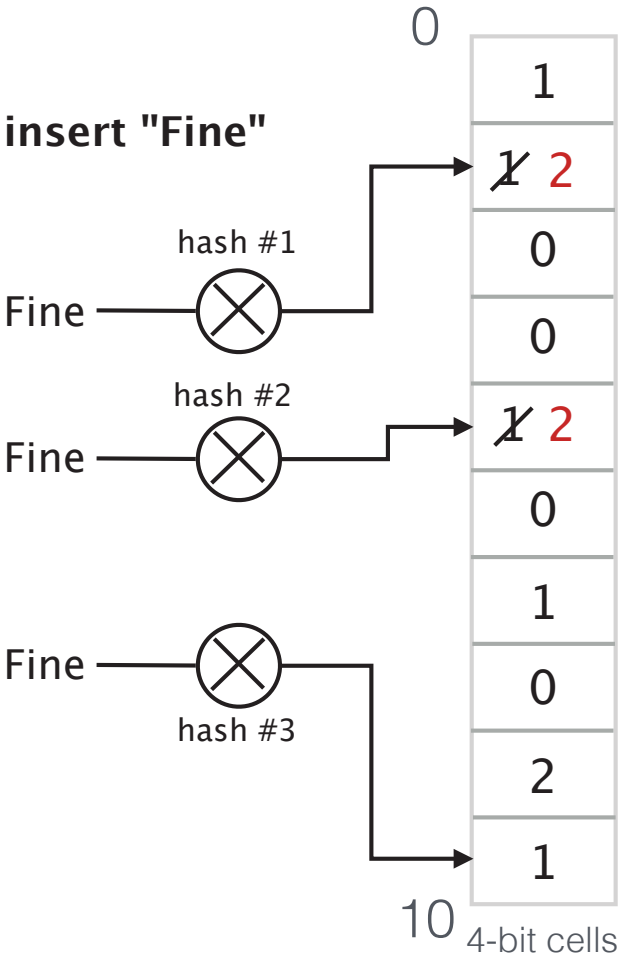
This extended version is called a Counting Bloom Filter



To add an element, increment the corresponding counters

But we can easily extend them to handle deletions

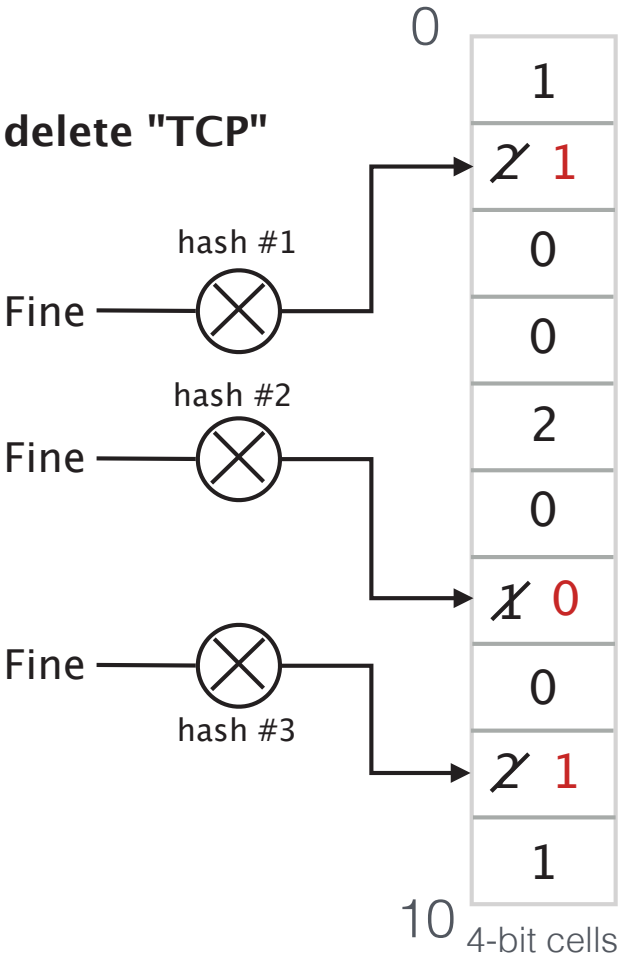
This extended version is called a Counting Bloom Filter



To add an element, increment the corresponding counters

But we can easily extend them to handle deletions

This extended version is called a Counting Bloom Filter

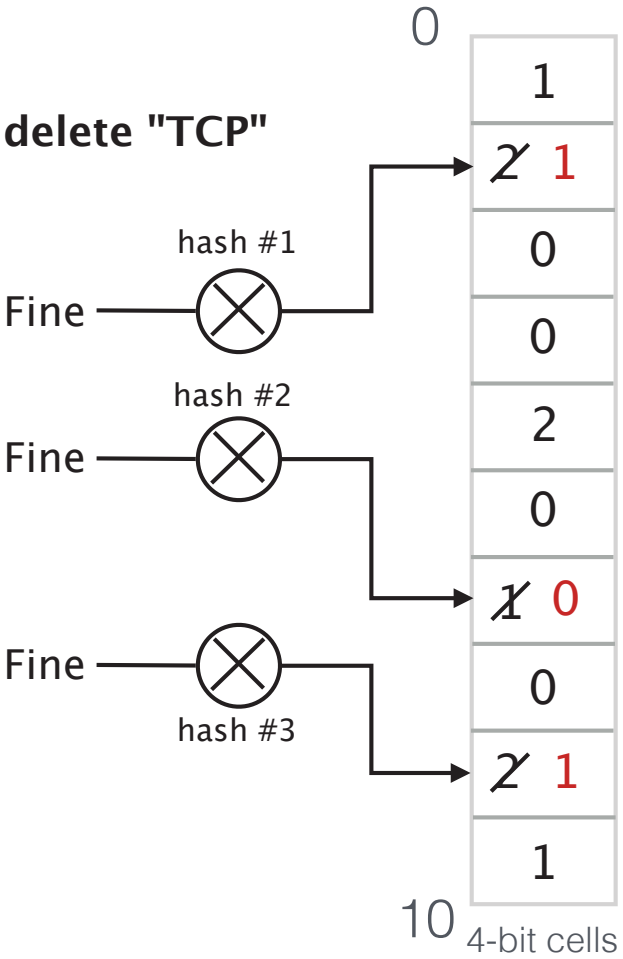


To add an element, increment the corresponding counters

To delete an element, decrement the corresponding counters

But we can easily extend them to handle deletions

This extended version is called a Counting Bloom Filter



To add an element, increment the corresponding counters

To delete an element, decrement the corresponding counters

All of our prior analysis for standard bloom filters applies to counting bloom filters

Counting Bloom Filters do handle deletions

at the price of using more memory

Counting Bloom Filters do handle deletions
at the price of using more memory

Counters must be large enough to avoid overflow
If a counter eventually overflows, the filter may yield
false negatives

Counting Bloom Filters do handle deletions
at the price of using more memory

Counters must be large enough to avoid overflow
If a counter eventually overflows, the filter may yield false negatives

Poisson approximation suggests 4 bits/counter
The average load (i.e., $\frac{NK}{M}$) is $\ln 2$ assuming $K = \ln 2 * (M/N)$
With N=10000 and M=80000 the probability that some counter overflows if we use b-bit counters is at most
$M * Pr(Poisson(\ln 2) \geq 2^b) = 1.78\text{e}{-11}$

Implementation of a Counting Bloom Filter in $P4_{16}$ with 2 hash functions

Add a new element

```
control MyIngress(…) {

  register register<bit<4>>(NB_CELLS) bloom_filter;

   apply {
      hash(meta.index1, HashAlgorithm.my_hash1, 0,
        {meta.dstPrefix, packet.ip.srcIP}, NB_CELLS);
      hash(meta.index2, HashAlgorithm.my_hash2, 0,
        {meta.dstPrefix, packet.ip.srcIP}, NB_CELLS);

      // Add a new element if not yet in the set
      bloom_filter.read(meta.query1, meta.index1);
      bloom_filter.read(meta.query2, meta.index2);

      if (meta.query1 == 0 || meta.query2 == 0) {
        bloom_filter.write(meta.index1, meta.query1 + 1);
        bloom_filter.write(meta.index2, meta.query2 + 1);
      }
  }
}
```

Implementation of a Counting Bloom Filter in $P4_{16}$ with 2 hash functions

Delete an element

```
control MyIngress(…) {

  register register<bit<32>>(NB_CELLS) bloom_filter;

   apply {
      hash(meta.index1, HashAlgorithm.my_hash1, 0,
        {meta.dstPrefix, packet.ip.srcIP}, NB_CELLS);
      hash(meta.index2, HashAlgorithm.my_hash2, 0,
        {meta.dstPrefix, packet.ip.srcIP}, NB_CELLS);

      // Delete a element only if it is in the set
      bloom_filter.read(meta.query1, meta.index1);
      bloom_filter.read(meta.query2, meta.index2);

      if (meta.query1 > 0 && meta.query2 > 0) {
        bloom_filter.write(meta.index1, meta.query1 - 1);
        bloom_filter.write(meta.index2, meta.query2 - 1);
      }
  }
}
```

So far we have seen how to do insertions, deletions and membership queries

	strategy #1	strategy #2
output	Deterministic	Probabilistic
number of required operations	Probabilistic	Deterministic

Bloom Filters

Counting Bloom Filters

Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing

Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing

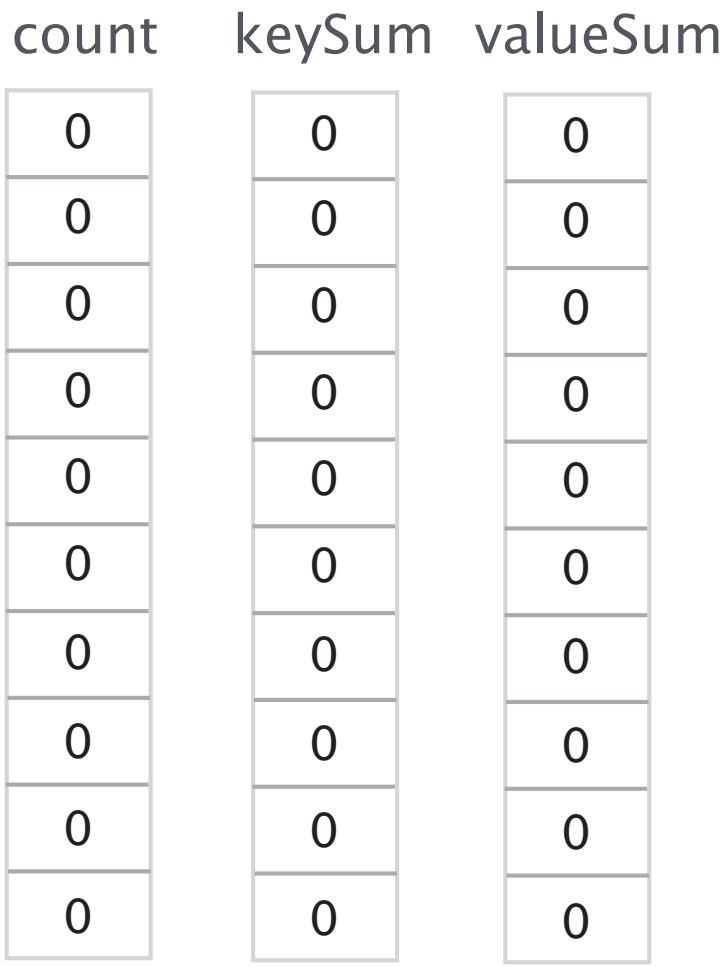
Each cell contains three fields

count which counts the number of entries mapped to this cell

keySum which is the sum of all the keys mapped to this cell

valueSum which is the sum of all the values mapped to this cell

Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing



Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing

Add a new key-value pair (assuming it is not in the set yet)

For each hash function
    hash the key to find the index

    Then at this index
        increment the count by one
        add key to keySum
        add value to valueSum

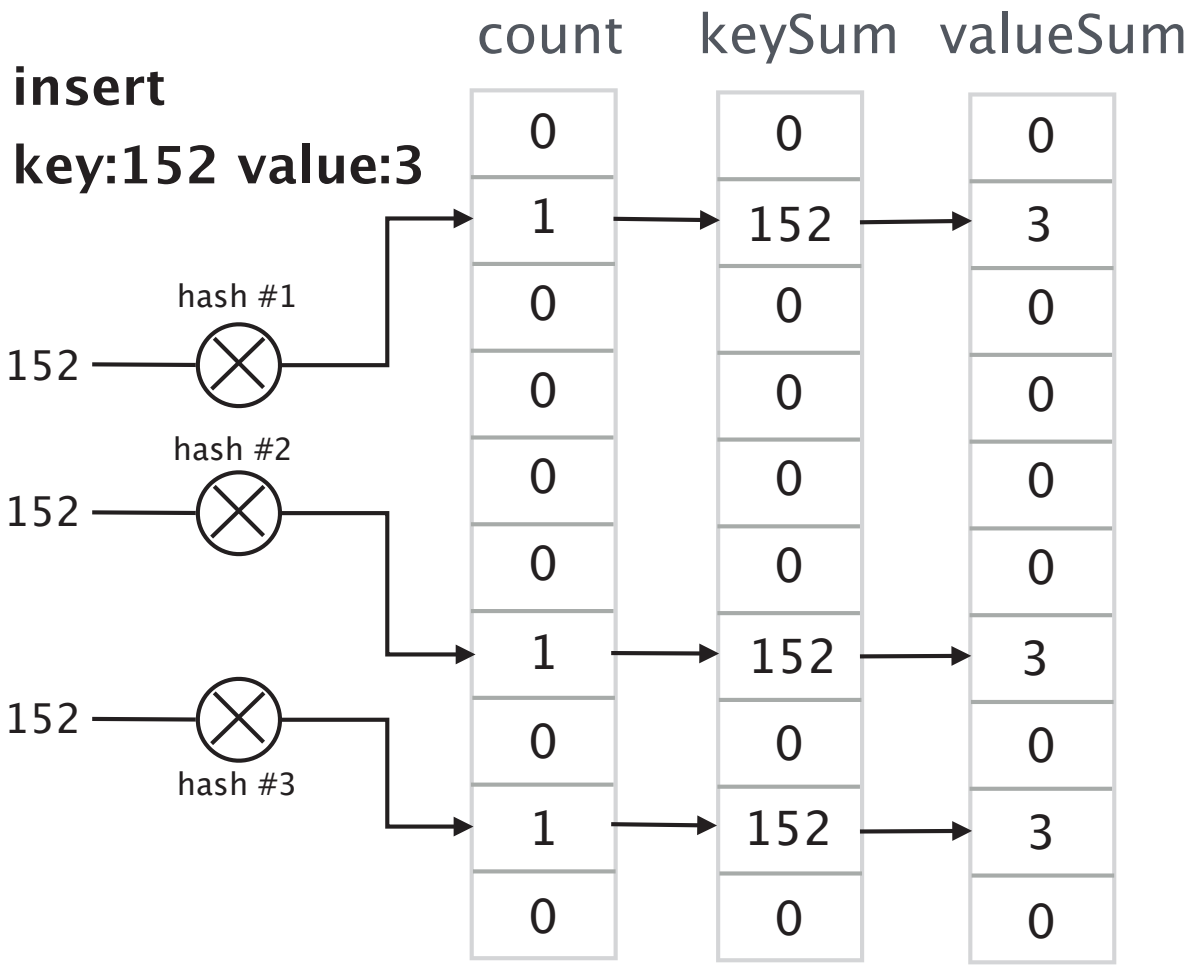
Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing

Delete a key-value pair (assuming it is in the set)

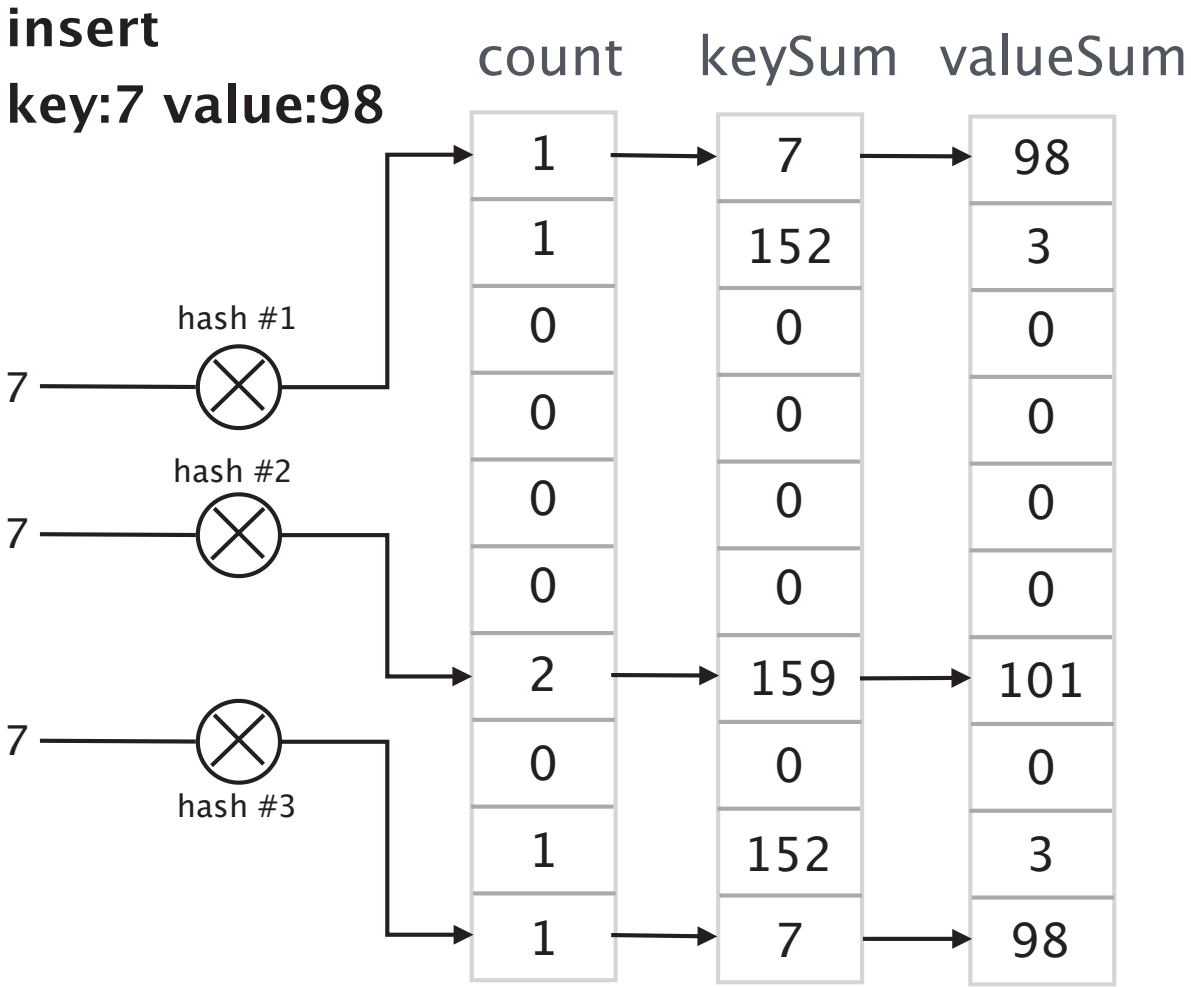
For each hash function
    hash the key to find the index

    Then at this index
        subtract one to the count
        subtract key to keySum
        subtract value to valueSum

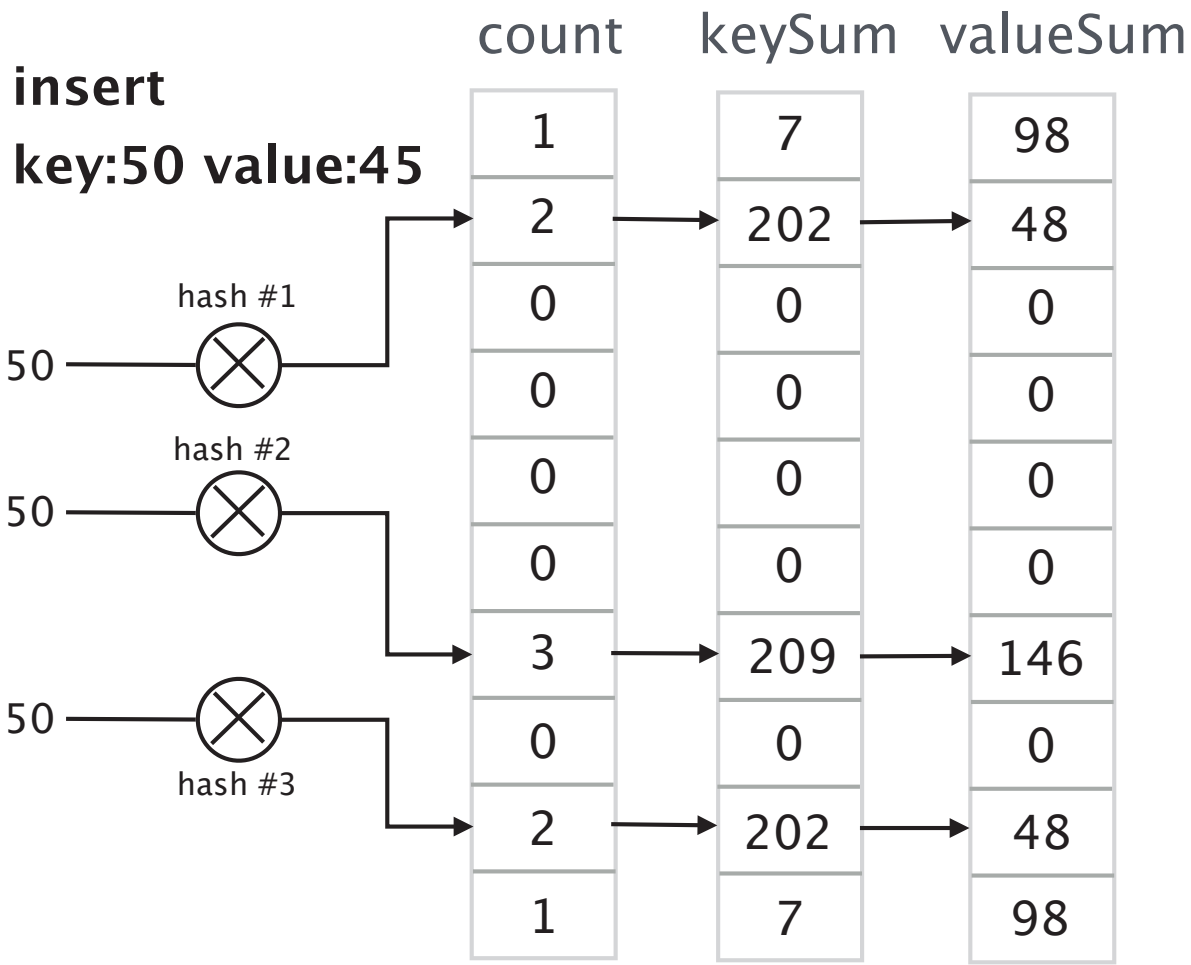
Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing



Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing



Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing

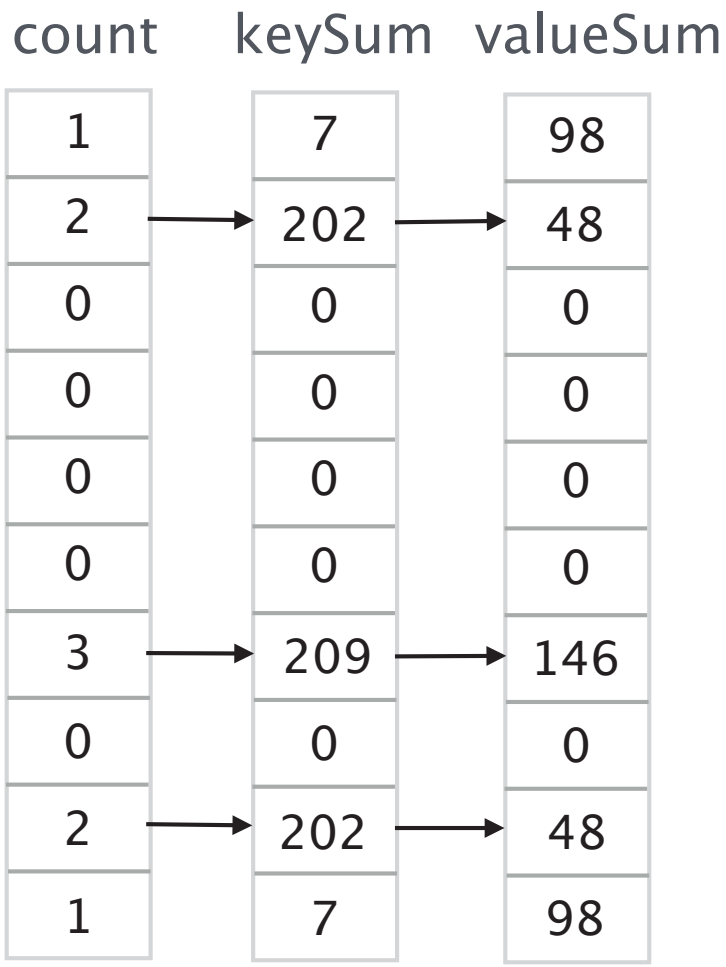


Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing

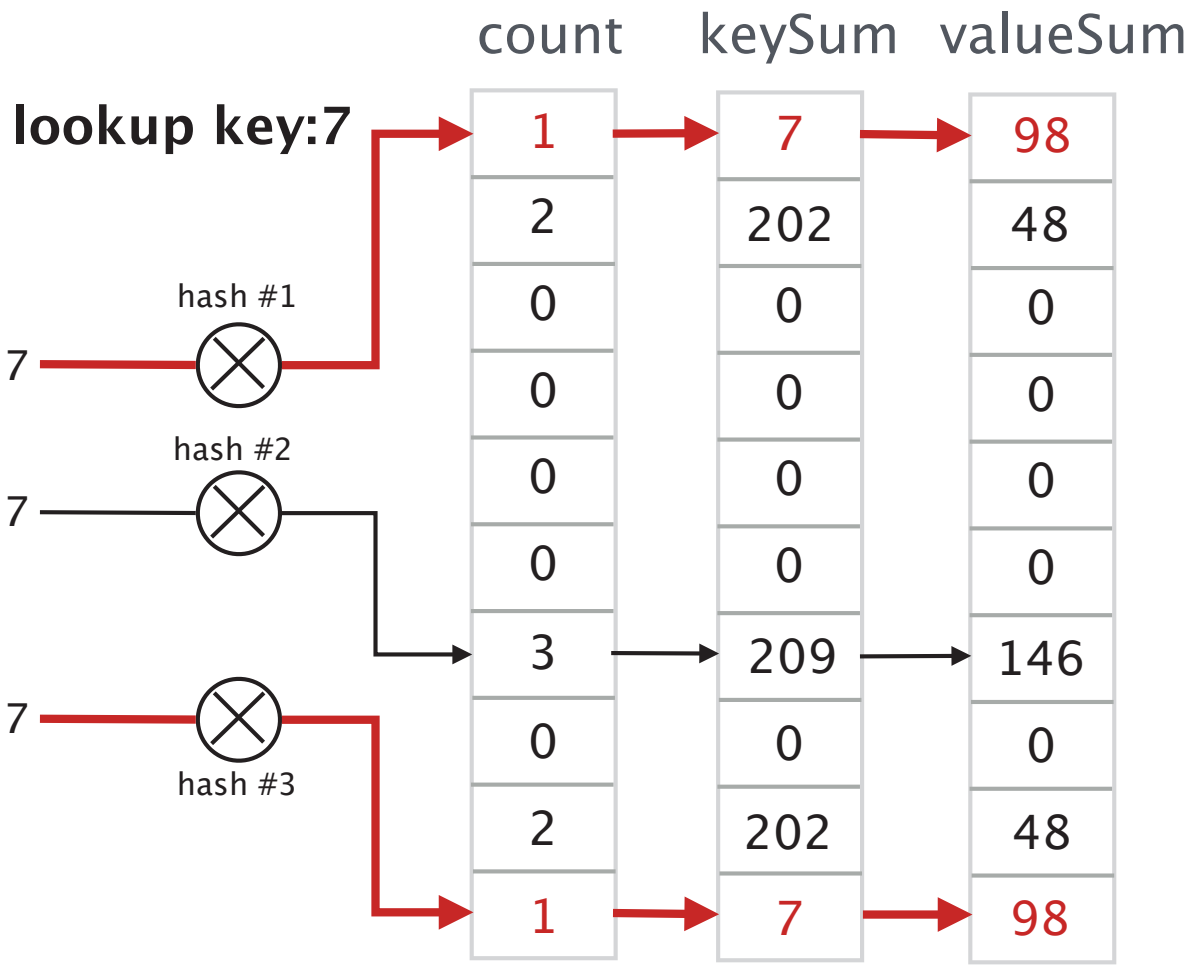
Key-value pair **lookup**

The value of a key can be found
if the key is associated to **at least**
one cell with a count = 1

Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing

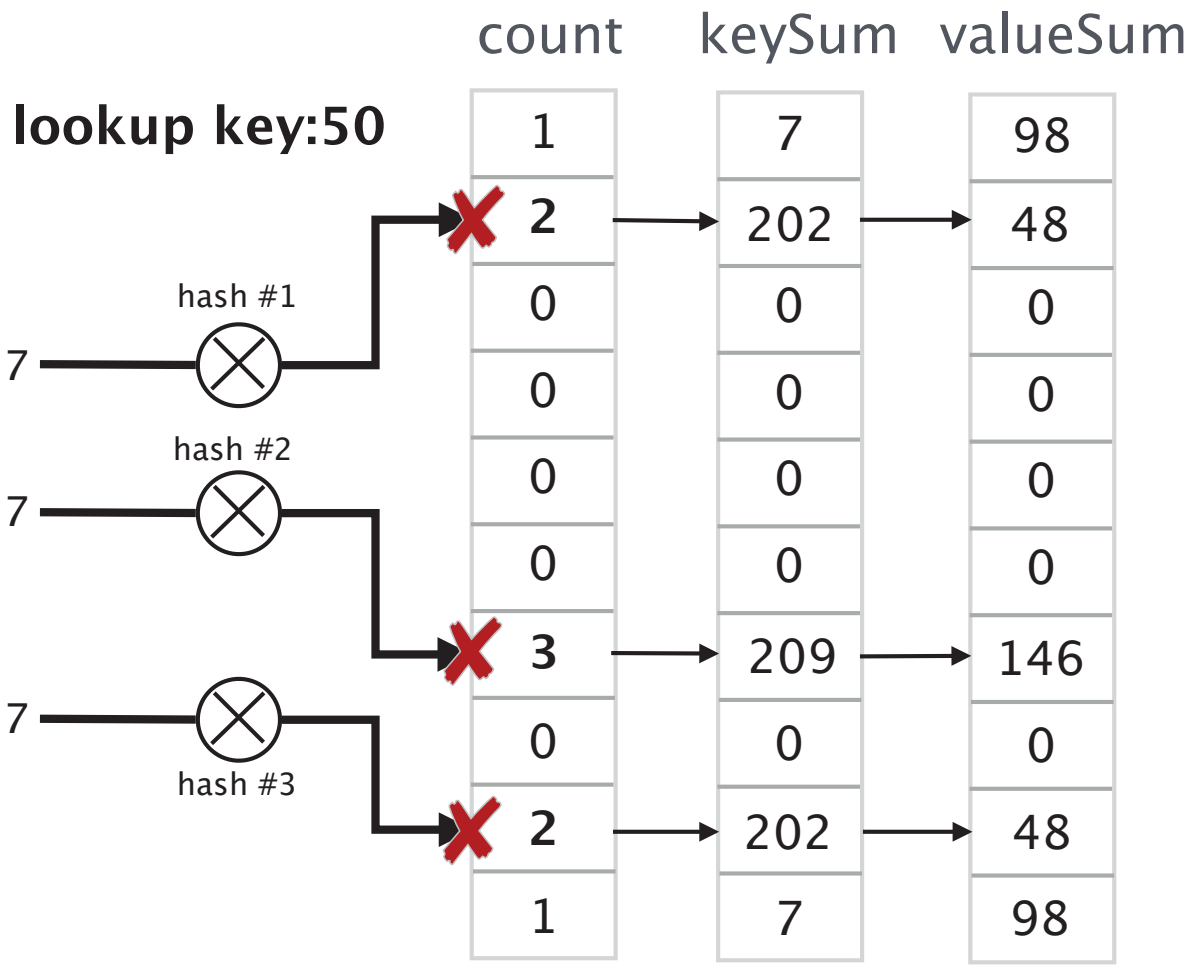


Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing



Key 7 has the value 98

Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing



The value for the key 50 can't be found

Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing

Listing the IBLT

While there is an index for which count = 1

    Find the corresponding key-value pair and return it

    Delete the corresponding key-value pair

Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing

Listing the IBLT

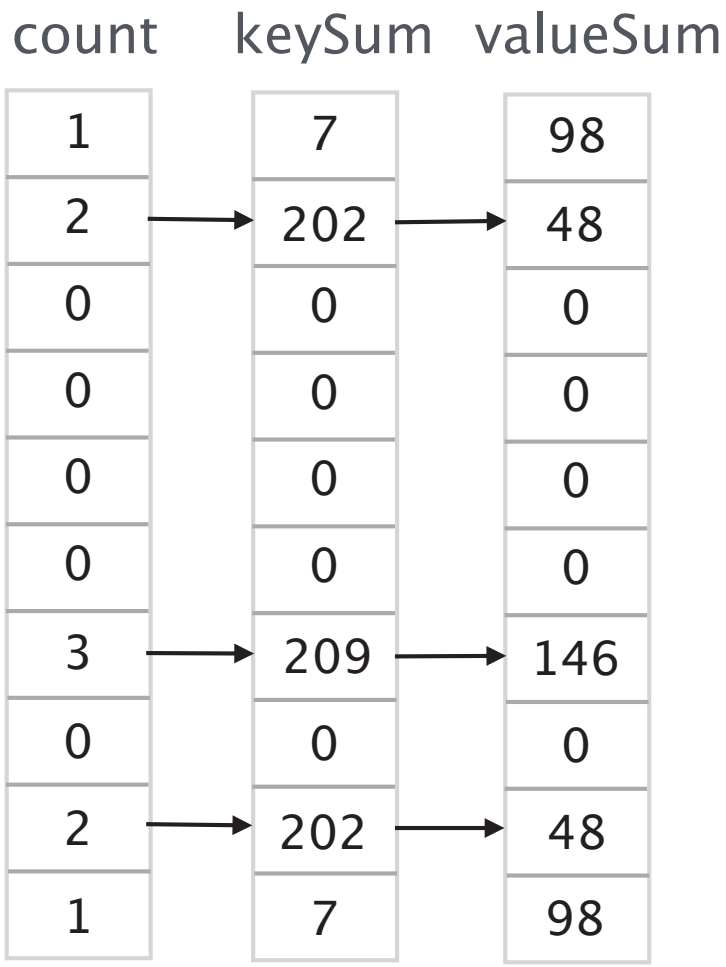
While there is an index for which count = 1

- **Find** the corresponding key-value pair and return it
- **Delete** the corresponding key-value pair

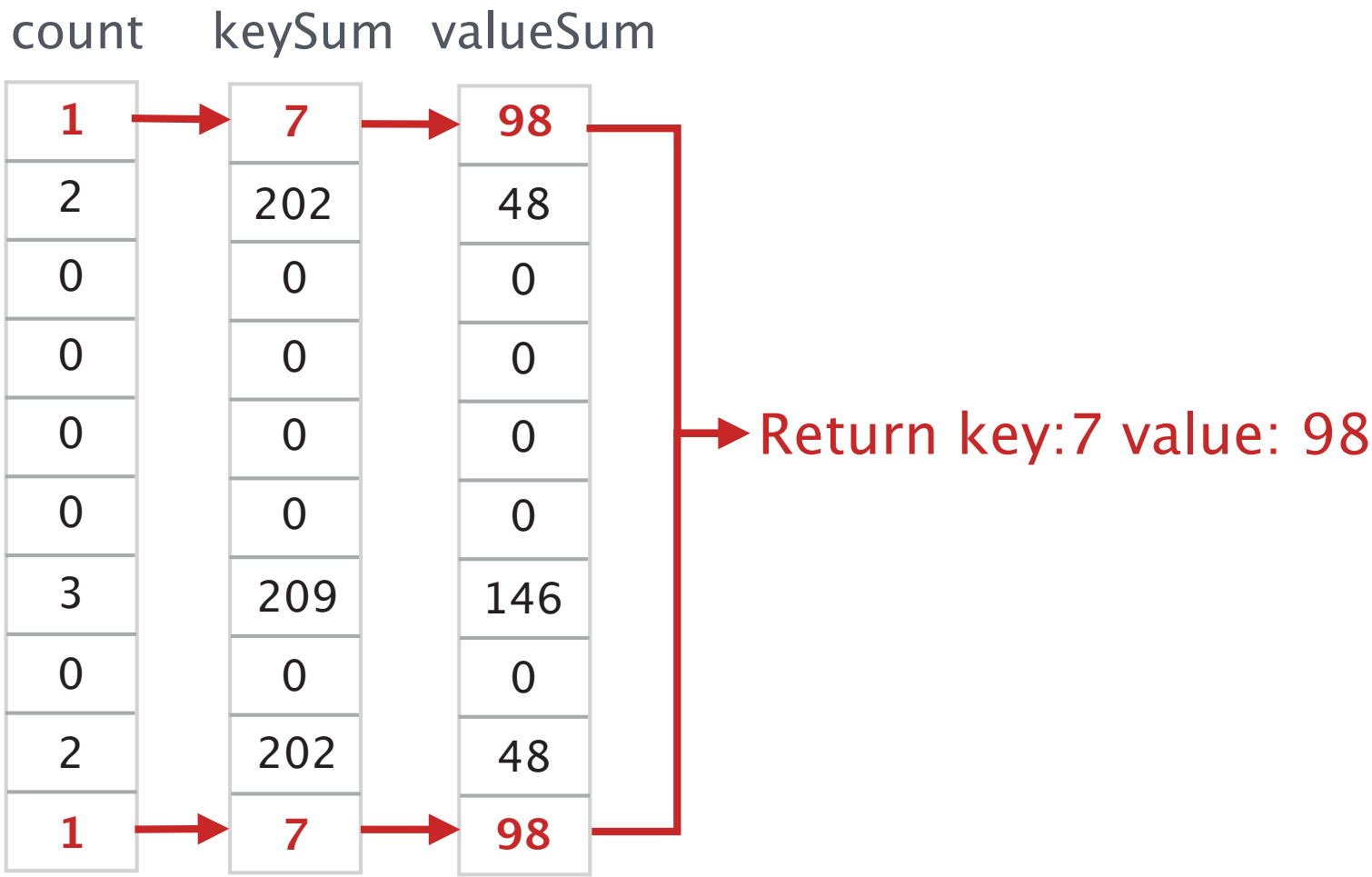
Unless the number of iterations is very low, loops can't be implemented in hardware

The listing is done by the controller

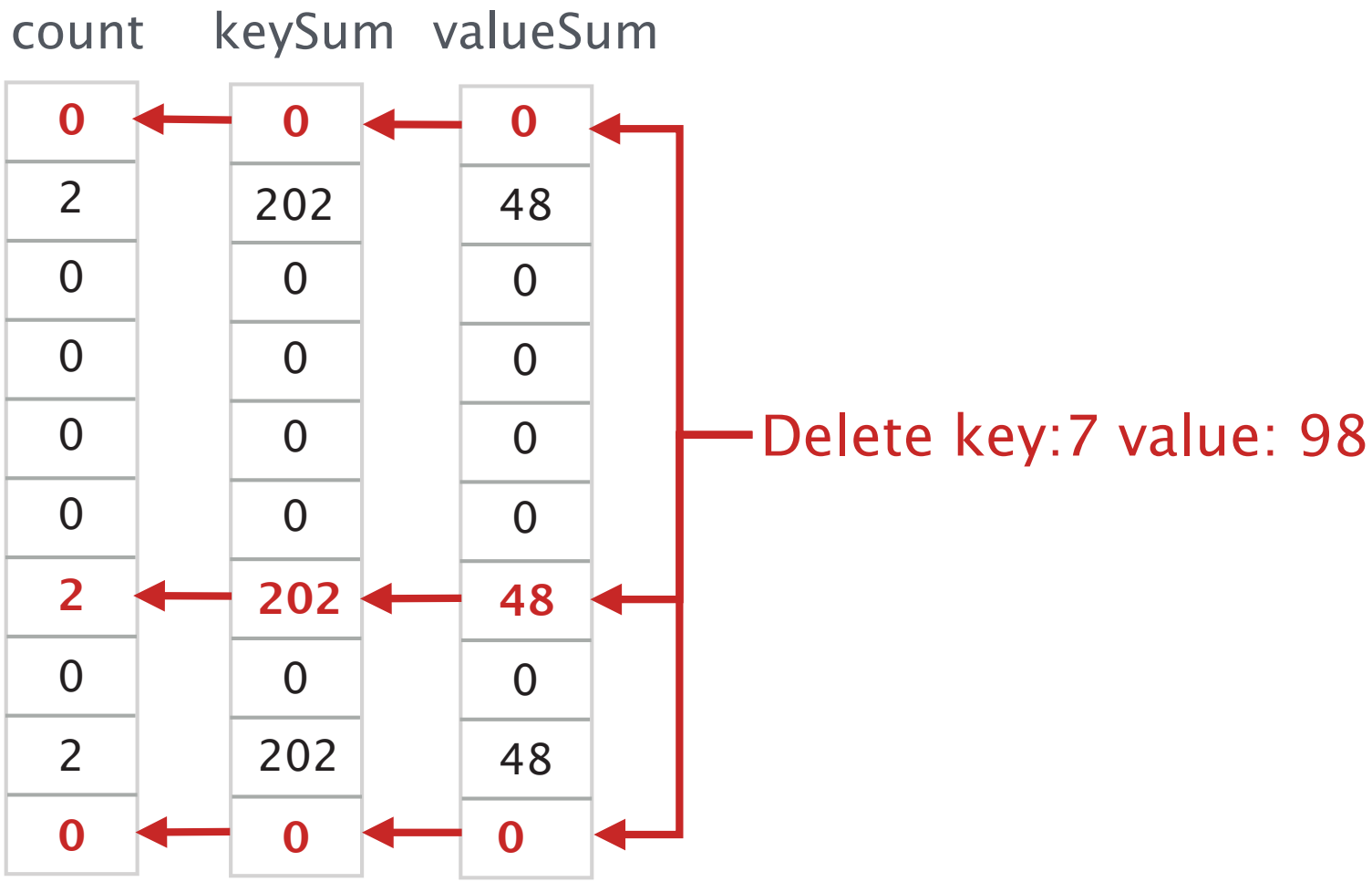
Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing



Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing



Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing



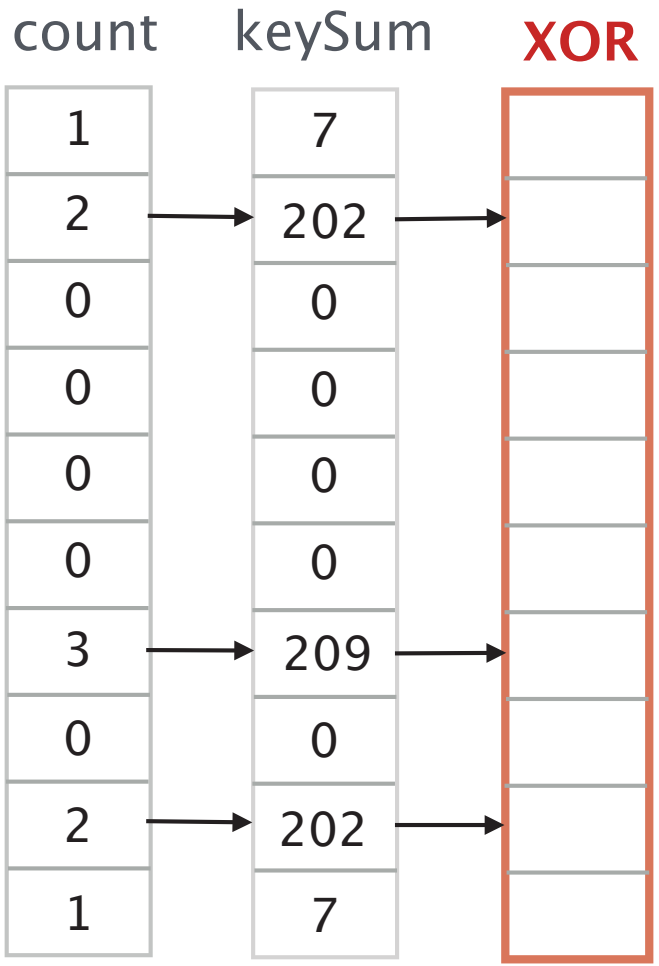
Invertible Bloom Lookup Tables (IBLT) stores key-value pairs and allows for lookups and a complete listing

count	keySum	valueSum
0	0	0
2	202	48
0	0	0
0	0	0
0	0	0
0	0	0
2	202	48
0	0	0
2	202	48
0	0	0

In this example, a complete listing is not possible

In many settings, we can use XORs in place of sums

For example to avoid overflow issues



For further information about Bloom Filters, Counting Bloom Filters and IBLT

Space/Time Trade-offs in Hash Coding
with Allowable Errors. Burton H. Bloom. 1970.

Network Applications of Bloom Filters: A Survey.
Andrei Broder and Michael Mitzenmacher. 2004.

Invertible Bloom Lookup Tables.
Michael T. Goodrich and Michael Mitzenmacher. 2015.

FlowRadar: A Better NetFlow for Data Centers
Yuliang Li et al. NSDI 2016.

Is a certain flow in the stream?

Bloom Filter

What flows are in the stream?

Invertible Bloom Filter, HyperLogLog Sketch, ...

How frequent does an flow appear?

Count Sketch, CountMin Sketch, ...

What are the most frequent elements?

Count/CountMin + Heap, ...

How many flows belong to a certain subnet?

SketchLearn *SigComm '18*

*In networking, we talk about **flows of packets**,*

but these questions apply to other domains as well,

*e.g. **search engines and databases**.*

Is a certain flow in the stream?

Bloom Filter

What flows are in the stream?

Invertible Bloom Filter, HyperLogLog Sketch, ...

How frequently does an flow appear?

Count Sketch, CountMin Sketch, ...

What are the most frequent elements?

Count/CountMin + Heap, ...

How many flows belong to a certain subnet?

SketchLearn *SIGCOMM '18*

We are going to look at **frequencies**,

i.e. **how often** an element occurs in a data stream.

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}$$

vector of frequencies (counts)

*of all **distinct elements** $\boldsymbol{x_i}$*

We are going to look at **frequencies**,

i.e. **how often** an element occurs in a data stream.

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}$$

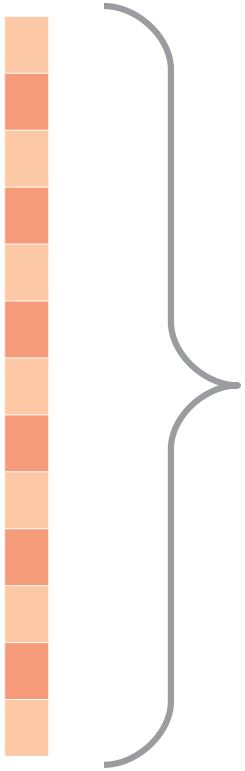
vector of frequencies (counts)

of all ***distinct elements*** $\boldsymbol{x_i}$

distinct flows

In the worst case, an algorithm providing **exact frequencies** requires **linear space**.

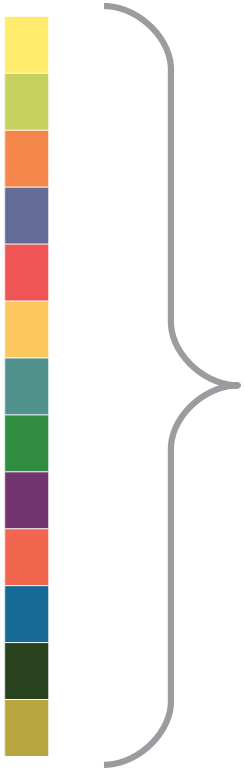
In the worst case, an algorithm providing **exact frequencies** requires **linear space**.



Data Stream

n elements in total

In the worst case, an algorithm providing **exact frequencies** requires **linear space**.



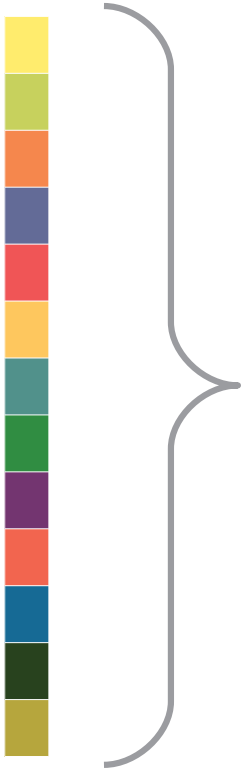
Data Stream

n elements in total

→ n **distinct elements**

(in the worst case)

In the worst case, an algorithm providing **exact frequencies** requires **linear space**.



Data Stream

n elements in total

→ n **distinct elements**

(in the worst case)

→ n **counters** required? **:(**

Probabilistic datastructures can help again!

Bloom Filters

quickly “filter” only those elements that might be in the set

Save space by allowing false positives.

Probabilistic datastructures can help again!

Bloom Filters

quickly "filter" only those elements that might be in the set

Save space by allowing false positives.

Sketches

provide a approximate frequencies of elemetns in a data stream.

Save space by allowing mis-counting.

Today we'll talk about: important questions,

how 'sketches' answer them,

limitations of 'sketches',

and my master thesis :)

*A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.*

*A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.*

Notation reminder:

*vector of frequencies (counts) of all **distinct elements x_i***

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}$$

*A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.*

$$Pr\left[\ \underbrace{\hat{x}_i}_{\substack{\textit{estimated}\\ \textit{frequency}}} - \underbrace{x_i}_{\substack{\textit{true}\\ \textit{frequency}}} \geq \varepsilon \underbrace{\|\boldsymbol{x}\|_1}_{\substack{\text{sum } \textit{of}\\ \textit{frequencies}}}\ \right] \leq \delta$$

The estimation error ***exceeds*** $\varepsilon\|\boldsymbol{x}\|_1$

with a ***probability smaller than*** δ

relative to L1 norm

$$Pr\left[\ \underbrace{\hat{x}_i}_{\substack{\textit{estimated}\\ \textit{frequency}}} - \underbrace{x_i}_{\substack{\textit{true}\\ \textit{frequency}}} \geq \varepsilon \underbrace{\|\boldsymbol{x}\|_1}_{\substack{\text{sum } \textit{of}\\ \textit{frequencies}}}\ \right] \leq \delta$$

The estimation error ***exceeds*** $\varepsilon \|\boldsymbol{x}\|_1$

with a ***probability smaller than*** δ

$$Pr\left[\ \underbrace{\hat{x}_i}_{\substack{\text{estimated}\\ \text{frequency}}} - \underbrace{x_i}_{\substack{\text{true}\\ \text{frequency}}} \geq \varepsilon \underbrace{\|\boldsymbol{x}\|_1}_{\substack{\text{sum of}\\ \text{frequencies}}}\ \right] \leq \delta$$

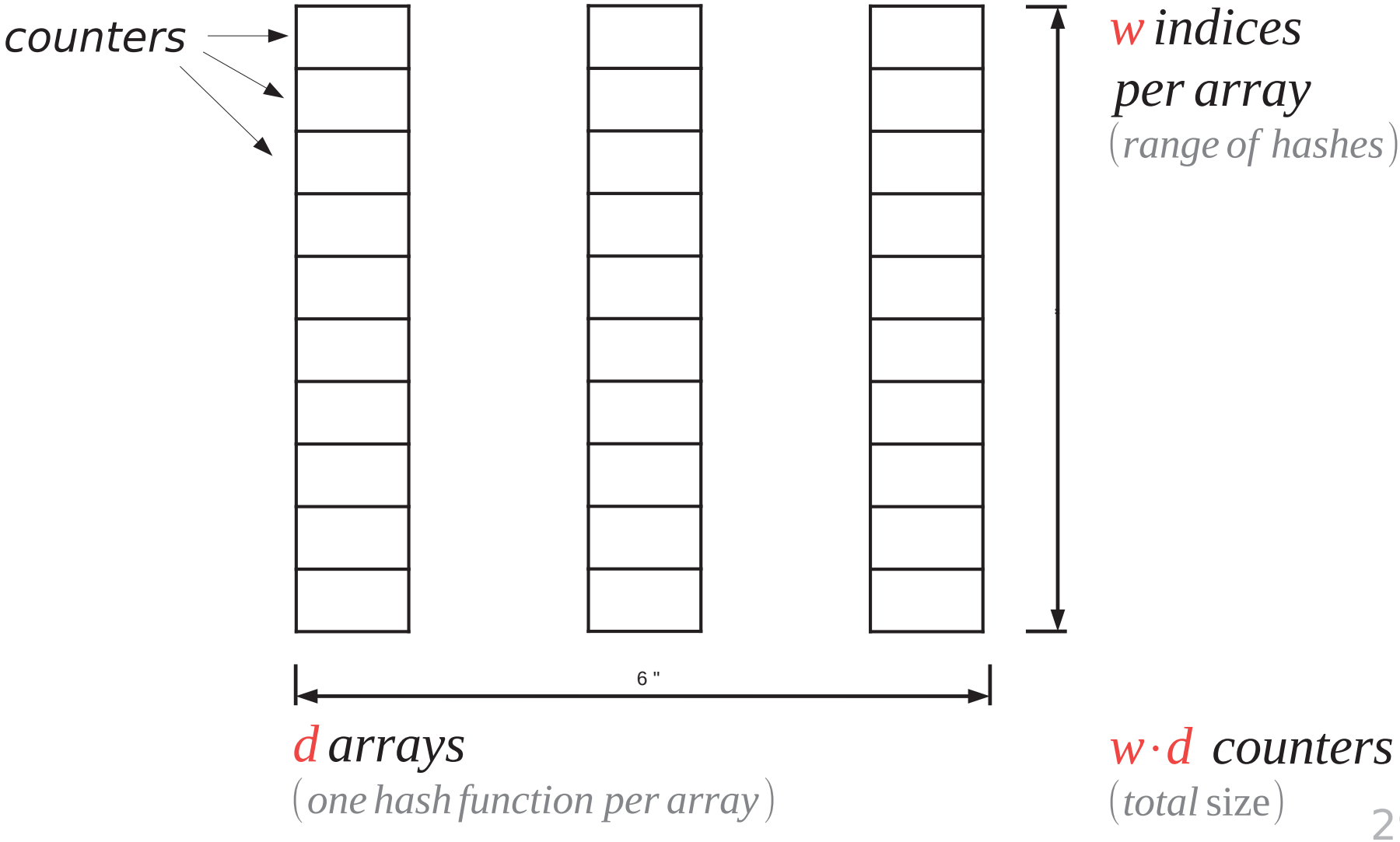
$$Let \quad \varepsilon = 0.01, \quad \delta = 0.05, \quad \|\boldsymbol{x}\|_1 = 10000$$

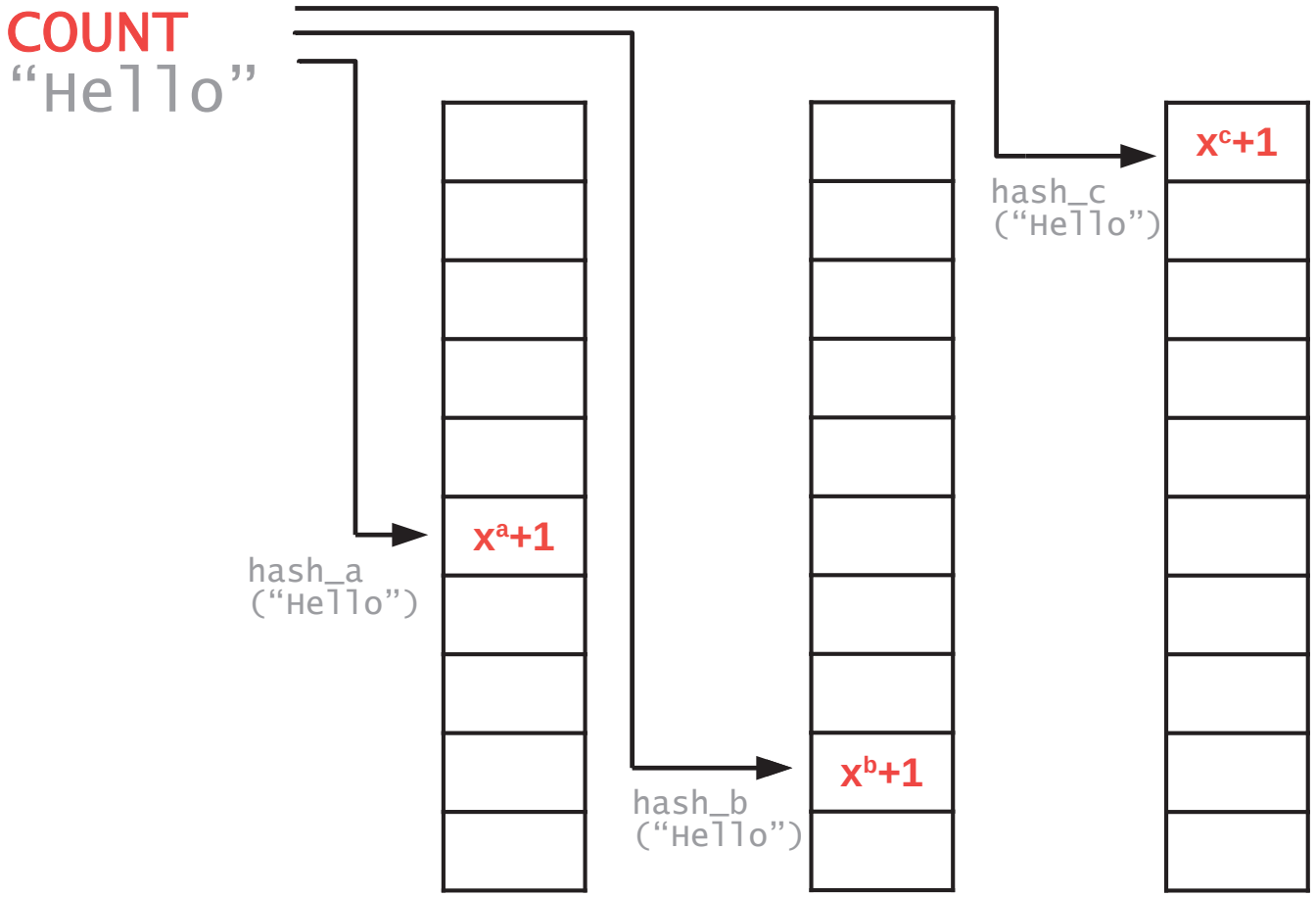
*The probability for **any estimate** to be off by **more than 100** is **less than 5%***

(after counting 10000 elements)

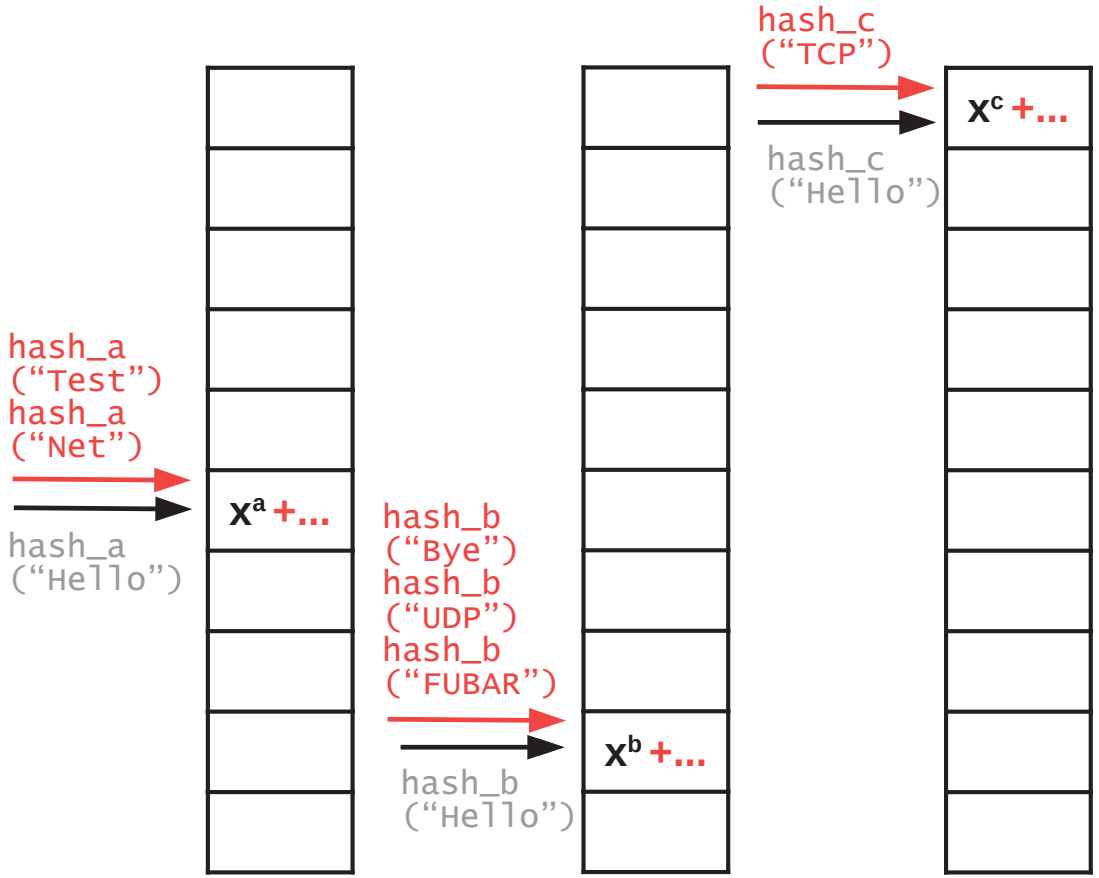
*A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.*

A **CountMin** Sketch uses multiple arrays and hashes.

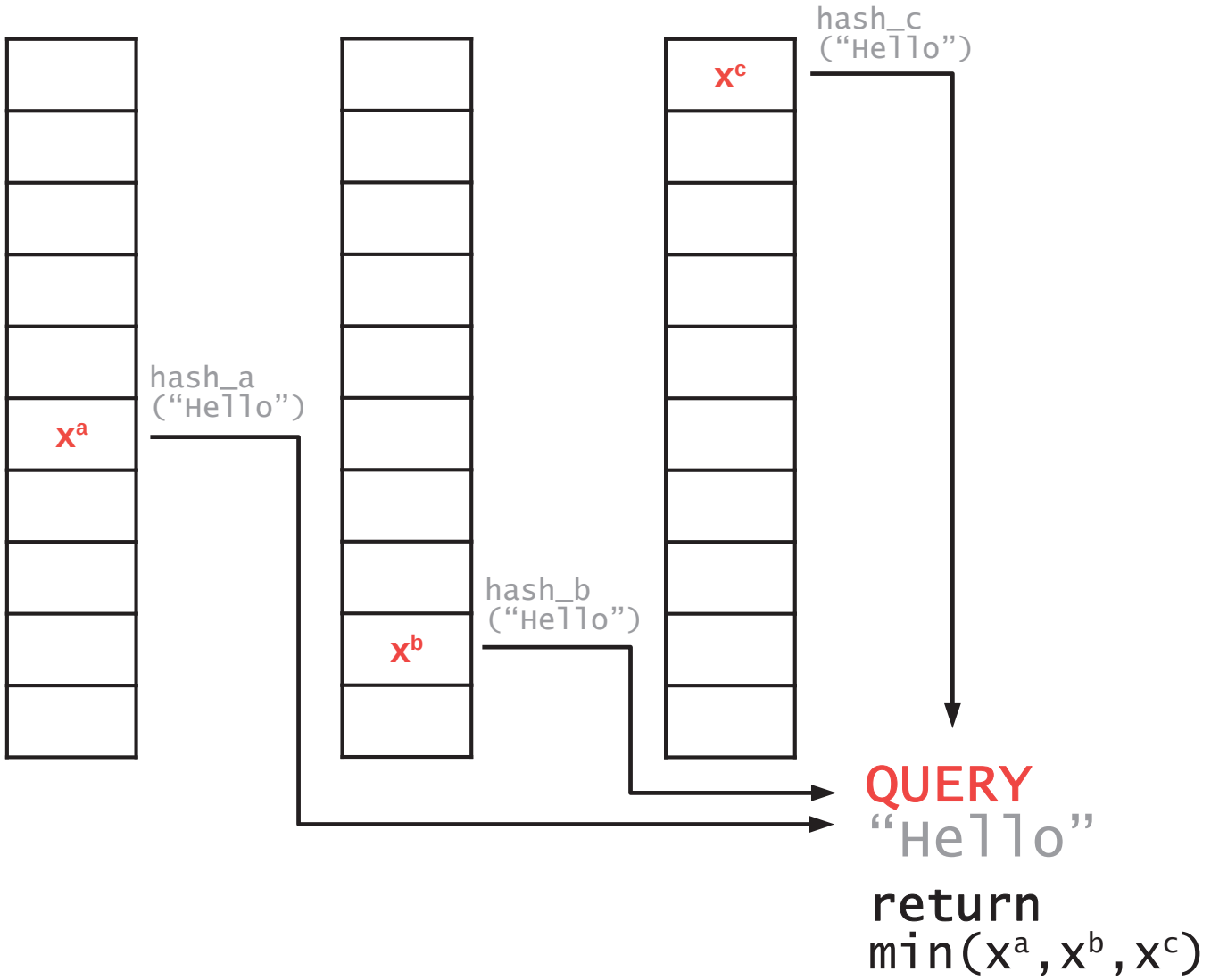



COUNT
“Hello”
hash_a
(“Hello”)
x^a+1
hash_b
(“Hello”)
x^b+1
hash_c
(“Hello”)
x^c+1

Hash collisions cause **over-counting**.



Returning the **minimum value** minimizes the error.



*A **CountMin sketch** uses the same principles as a counting bloom filter, but is designed to have **provable** **L1 error bounds** for frequency queries.*

$$Pr\left[\ \underbrace{\hat{x}_i}_{\substack{\textit{estimated}\\ \textit{frequency}}} - \underbrace{x_i}_{\substack{\textit{true}\\ \textit{frequency}}} \geq \varepsilon \underbrace{\|\boldsymbol{x}\|_1}_{\substack{\text{sum } \textit{of}\\ \textit{frequencies}}}\ \right] \leq \delta$$

Understanding the error bounds allows **dimensioning** the sketch optimally.

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$\hat{x}_i = \min_{h \in h_1 .. h_d} \hat{x}_i^h$$

estimated frequency

estimate for specific hash

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$\hat{x}_i = \min_{h \in h_1 .. h_d} \hat{x}_i^h$$

estimated frequency

estimate for specific hash

The error bounds can be derived with **Markov's Inequality**

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

$$\Pr[X \geq c \cdot E[X]] \leq \frac{1}{c}$$

wikipedia.org/wiki/Markov's_inequality

The error bounds can be derived with **Markov's Inequality**

Error Bounds
per hash/array

$$\Pr\left[\hat{x}_i^h - x_i \geq c \cdot E\left[\hat{x}_i^h - x_i\right]\right] \leq \frac{1}{c}$$

Error Bounds
for the minimum

Optimal Size

wikipedia.org/wiki/Markov's_inequality

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq c \cdot E\left[\hat{x}_i^h - x_i\right]\right] \leq \frac{1}{c}$$

$$\hat{x}_i^h = x_i + \sum_{x_j \neq x_i} x_j 1_h(x_i, x_j)$$

true frequency

over-counting from hash collisions

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq c \cdot E\left[\hat{x}_i^h - x_i\right]\right] \leq \frac{1}{c}$$

$$\hat{x}_i^h = x_i + \sum_{x_j \neq x_i} x_j \, 1_h(x_i, x_j)$$

hash collision

$$= \begin{cases} 1, & \text{if } h(x_i) = h(x_j) \\ 0, & \text{otherwise} \end{cases}$$

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq c \cdot E\left[\hat{x}_i^h - x_i\right]\right] \leq \frac{1}{c}$$

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \, 1_h\left(x_i, x_j\right)$$

estimation error

over-counting from hash collisions

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq c \cdot E\left[\hat{x}_i^h - x_i\right]\right] \leq \frac{1}{c}$$

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \, 1_h\left(x_i, x_j\right)$$

$$E\left[\hat{x}_i^h - x_i\right] = E\left[\sum_{x_j \neq x_i} x_j \, 1_h\left(x_i, x_j\right)\right]$$

We treat the **data as a constant** and the **hash as a random function** with certain properties.

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq c \cdot E\left[\hat{x}_i^h - x_i\right]\right] \leq \frac{1}{c}$$

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \, 1_h(x_i, x_j)$$

$$E\left[\hat{x}_i^h - x_i\right] = E\left[\sum_{x_j \neq x_i} x_j \, 1_h(x_i, x_j)\right]$$

constant *random*

wikipedia.org/wiki/Universal_hashing

We treat the **data as a constant** and the **hash as a random function** with certain properties.

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq c \cdot E\left[\hat{x}_i^h - x_i\right]\right] \leq \frac{1}{c}$$

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \, 1_h(x_i, x_j)$$

$$E\left[\hat{x}_i^h - x_i\right] = \sum_{x_j \neq x_i} x_j \, E\left[1_h(x_i, x_j)\right]$$

wikipedia.org/wiki/Universal_hashing

We treat the **data as a constant** and the **hash as a random function** with certain properties.

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq c \cdot E\left[\hat{x}_i^h - x_i\right]\right] \leq \frac{1}{c}$$

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \, 1_h\left(x_i, x_j\right)$$

$$E\left[\hat{x}_i^h - x_i\right] = \sum_{x_j \neq x_i} x_j \underbrace{E\left[1_h\left(x_i, x_j\right)\right]}_{\leq \frac{1}{w}}$$

wikipedia.org/wiki/Universal_hashing

We treat the **data as a constant** and the **hash as a random function** with certain properties.

Error Bounds
per hash/array

$$\Pr\left[\hat{x}_i^h - x_i \geq c \cdot E\left[\hat{x}_i^h - x_i\right]\right] \leq \frac{1}{c}$$

Error Bounds
for the minimum

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \, 1_h\left(x_i, x_j\right)$$

Optimal Size

$$\mathrm{E}\left[\hat{x}_i^h - x_i\right] \leq \sum_{x_j \neq x_i} x_j \frac{1}{w}$$

wikipedia.org/wiki/Universal_hashing

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq c \cdot E\left[\hat{x}_i^h - x_i\right]\right] \leq \frac{1}{c}$$

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \, 1_h\left(x_i, x_j\right)$$

$$\mathrm{E}\left[\hat{x}_i^h - x_i\right] \leq \sum_{x_j \neq x_i} x_j \frac{1}{w} \leq \sum_{x_j} x_j \frac{1}{w}$$

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq c \cdot E\left[\hat{x}_i^h - x_i\right]\right] \leq \frac{1}{c}$$

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j 1_h\left(x_i, x_j\right)$$

$$\mathrm{E}\left[\hat{x}_i^h - x_i\right] \leq \sum_{x_j \neq x_i} x_j \frac{1}{w} \leq \|\boldsymbol{x}\|_1 \frac{1}{w}$$

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq c \cdot \underbrace{E\left[\hat{x}_i^h - x_i\right]}_{\leq \frac{1}{w}\|\boldsymbol{x}\|_1}\right] \leq \frac{1}{c}$$

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq \frac{c}{w}\left\|\boldsymbol{x}\right\|_1\right] \leq \frac{1}{c}$$

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq \underbrace{\varepsilon^h}_{\frac{c}{w}} \|\boldsymbol{x}\|_1\right] \leq \underbrace{\delta^h}_{\frac{1}{c}}$$

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq \underbrace{\varepsilon^h}_{\frac{c}{w}} \|\boldsymbol{x}\|_1\right] \leq \underbrace{\delta^h}_{\frac{1}{c}}$$

*The **estimate for each hash** has a well defined **L1 error bound**.*

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

$$\Pr\left[\hat{x}_i^h - x_i \geq \underbrace{\varepsilon^h}_{\frac{c}{w}} \|\boldsymbol{x}\|_1\right] \leq \underbrace{\delta^h}_{\frac{1}{c}}$$

*The **estimate for each hash** has a well defined **L1 error bound**.*

What about the minimum?

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$Pr\left[\hat{x}_i - x_i \geq \frac{c}{w}\left\|\boldsymbol{x}\right\|_1\right] \quad \leq \quad ?$$

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$Pr\left[\underbrace{\min_{h \in h_1 .. h_d} \hat{x}_i^h}_{\hat{x}_i} - x_i \geq \frac{c}{w} \|\boldsymbol{x}\|_1\right] \leq \textbf{?}$$

Multiple hash functions work like **independent trials**.

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

$$Pr\Big[\underbrace{\min_{h \in h_1 .. h_d} \hat{x}_i^h}_{\hat{x}_i} - x_i \geq \frac{c}{w}\|\boldsymbol{x}\|_1\Big] \leq \textbf{?}$$

$$\Leftrightarrow$$

$$\prod_{h \in h_1 .. h_d} Pr\left[\hat{x}_i^h - x_i \geq \frac{c}{w}\|\boldsymbol{x}\|_1\right] \leq \textbf{?}$$

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$Pr\Big[\underbrace{\min_{h \in h_1 .. h_d} \hat{x}_i^h}_{\hat{x}_i} - x_i \geq \frac{c}{w} \|\boldsymbol{x}\|_1\Big] \leq \textbf{?}$$

$$\Leftrightarrow$$

$$\prod_{h \in h_1 .. h_d} \underbrace{Pr\left[\hat{x}_i^h - x_i \geq \frac{c}{w} \|\boldsymbol{x}\|_1\right]}_{\leq \frac{1}{c}} \leq \textbf{?}$$

error bound per hash

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

$$Pr\Big[\underbrace{\min_{h \in h_1 .. h_d} \hat{x}_i^h}_{\hat{x}_i} - x_i \geq \frac{c}{w} \|\boldsymbol{x}\|_1\Big] \leq \; ?$$

$$\Leftrightarrow$$

$$\prod_{h \in h_1 .. h_d} \underbrace{Pr\left[\hat{x}_i^h - x_i \geq \frac{c}{w} \|\boldsymbol{x}\|_1\right]}_{\leq \frac{1}{c}} \leq \frac{1}{c^d}$$

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$Pr\left[\underbrace{\min_{h \in h_1 .. h_d} \hat{x}_i^h}_{\hat{x}_i} - x_i \geq \frac{c}{w} \|\boldsymbol{x}\|_1\right] \leq \frac{1}{c^d}$$

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$Pr\left[\hat{x}_i - x_i \geq \frac{c}{w}\left\|\boldsymbol{x}\right\|_1\right] \leq \frac{1}{c^d}$$

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$Pr\left[\hat{x}_i - x_i \geq \underbrace{\varepsilon}_{\frac{c}{w}} \|\boldsymbol{x}\|_1\right] \leq \underbrace{\delta}_{\frac{1}{c^d}}$$

We have proven the error bounds!

But what about the constant c?

For **every c**, there is a pair (d, w) achieving the error bound and confidence (ε, δ).

Error Bounds
per hash/array

$$\varepsilon = \frac{c}{w} \quad \Rightarrow \quad w = \left\lceil \frac{c}{\varepsilon} \right\rceil \qquad \textit{(hash range)}$$

Error Bounds
for the minimum

$$\delta = \frac{1}{c^d} \quad \Rightarrow \quad d = \left\lceil \log_c \frac{1}{\delta} \right\rceil \qquad \textit{(\#hashes)}$$

Optimal Size

Choosing c=e **minimizes** the total **number of counters**.

Error Bounds
per hash/array

$$\varepsilon = \frac{e}{w} \quad \Rightarrow \quad w = \left\lceil \frac{e}{\varepsilon} \right\rceil \qquad \textit{(hash range)}$$

$$\delta = \frac{1}{e^d} \quad \Rightarrow \quad d = \left\lceil \ln \frac{1}{\delta} \right\rceil \qquad \textit{(\#hashes)}$$

Error Bounds
for the minimum

$$d \cdot w = \frac{c}{\varepsilon} \log_c \frac{1}{\delta} \overset{\textit{minimize}}{=} \frac{e}{\varepsilon} \ln \frac{1}{\delta}$$

Optimal Size

A **CountMin** sketch recipe

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

Given ε, δ***, choosing***

$$w = \left\lceil \frac{e}{\varepsilon} \right\rceil$$ *(hash range)*

$$d = \left\lceil \ln \frac{1}{\delta} \right\rceil$$ *(#hashes)*

requires the ***minimum number of counters*** *s.t. the CountMin Sketch can guarantee that*

$$\hat{x}_i - x_i \geq \varepsilon \left\| \boldsymbol{x} \right\|_1$$

with a probability less than δ

*A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.*

*A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.*

***CountMin** sketch recipe*

Choose $d = \left\lceil \ln \frac{1}{\delta} \right\rceil, w = \left\lceil \frac{e}{\varepsilon} \right\rceil$

Then $\hat{x}_i - x_i \geq \varepsilon \|\boldsymbol{x}\|_1$ *with a probability less than* δ

*A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.*

→ only one design out of many!

*A **Count sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L2 error bounds** for frequency queries.*

The Count sketch uses **additional hashing** to give **L2 error bounds**, but requires more **resources**.

CountMin sketch

$h_1, \ldots, h_d$: $U \to \{1, \ldots, w\}$

COUNT x_i:

for h in $h_1, \ldots, h_d$:

$Reg_h[h(x_i)] + 1$

QUERY x_i:

return $\min_{h \text{ in } h1, \ldots, hd}$(

  $Reg_h[h(x_i)]$

)

The Count sketch uses **additional hashing** to give **L2 error bounds**, but requires more **resources**.

CountMin sketch

$h_1, \ldots, h_d$: $U \to \{1, \ldots, w\}$

COUNT x_i:

for h in $h_1, \ldots, h_d$:

$Reg_h[h(x_i)] + 1$

QUERY x_i:

return $\min_{h \text{ in } h1, \ldots, hd}$(

  $Reg_h[h(x_i)]$

)

Count sketch

$h_1, \ldots, h_d$: $U \to \{1, \ldots, w\}$

g: $U \to \{+1, -1\}$

COUNT x_i:

for h in $h_1, \ldots, h_d$:

$Reg_h[h(x_i)] + g(x_i)$

QUERY x_i:

return $\text{median}_{h \text{ in } h1, \ldots, hd}$ (

  $Reg_h[h(x_i)] * g(x_i)$

)

The Count sketch uses **additional hashing** to give **L2 error bounds**, but requires more **resources**.

***CountMin** sketch recipe*

Choose $d = \left\lceil \ln \frac{1}{\delta} \right\rceil, w = \left\lceil \frac{e}{\varepsilon} \right\rceil$

Then $\hat{x}_i - x_i \geq \varepsilon \|\mathbf{x}\|_1$ *with a probability less than* δ

The Count sketch uses **additional hashing** to give **L2 error bounds**, but requires more **resources**.

***CountMin** sketch recipe*

Choose $d = \left\lceil \ln \frac{1}{\delta} \right\rceil, w = \left\lceil \frac{e}{\varepsilon} \right\rceil$

Then $\hat{x}_i - x_i \geq \varepsilon \|\mathbf{x}\|_1$ *with a probability less than* δ

***Count** sketch recipe*

Choose $d = \left\lceil \ln \frac{1}{\delta} \right\rceil, w = \left\lceil \frac{e}{\varepsilon^2} \right\rceil$

Then $\hat{x}_i - x_i \geq \varepsilon \|\mathbf{x}\|_2$ *with a probability less than* δ

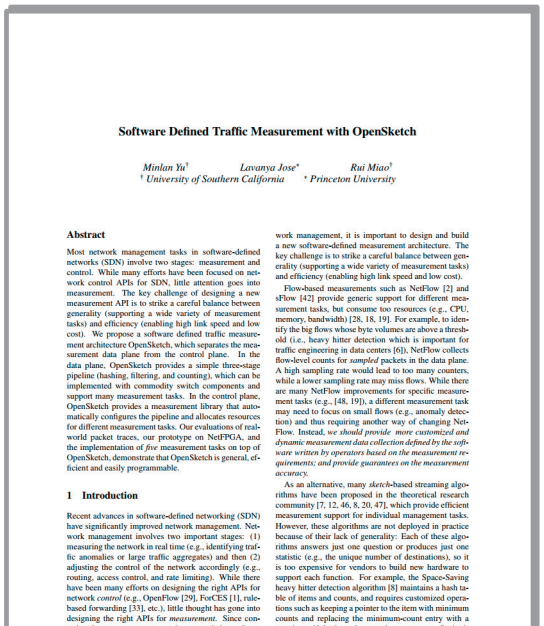
Sketches **are the new black**

...and many more!

OpenSketch

NSDI ‘13

[source]



Software Defined Traffic Measurement with OpenSketch

Minlan Yu† Lavanya Jose* Rui Miao†
† University of Southern California * Princeton University

Abstract

Most network management tasks in software-defined networks (SDN) involve two stages: measurement and control. While many efforts have been focused on network control APIs for SDN, little attention goes into measurement. The key challenge of designing a new measurement API is to strike a careful balance between generality (supporting a wide variety of measurement tasks) and efficiency (enabling high link speed and low cost). We propose a software defined traffic measurement architecture OpenSketch, which separates the measurement data plane from the control plane. In the data plane, OpenSketch provides a simple three-stage pipeline (hashing, filtering, and counting), which can be implemented with commodity switch components and support many measurement tasks. In the control plane, OpenSketch provides a measurement library that automatically configures the pipeline and allocates resources for different measurement tasks. Our evaluations of real-world packet traces, our prototype on NetFPGA, and the implementation of *five* measurement tasks on top of OpenSketch, demonstrate that OpenSketch is general, efficient and easily programmable.

1 Introduction

Recent advances in software-defined networking (SDN) have significantly improved network management. Network management involves two important stages: (1) measuring the network in real time (e.g., identifying traffic anomalies or large traffic aggregates) and then (2) adjusting the control of the network accordingly (e.g., routing, access control, and rate limiting). While there have been many efforts on designing the right APIs for network *control* (e.g., OpenFlow [29], ForCES [1], rule-based forwarding [33], etc.), little thought has gone into designing the right APIs for *measurement*. Since network management, it is important to design and build a new software-defined measurement architecture. The key challenge is to strike a careful balance between generality (supporting a wide variety of measurement tasks) and efficiency (enabling high link speed and low cost).

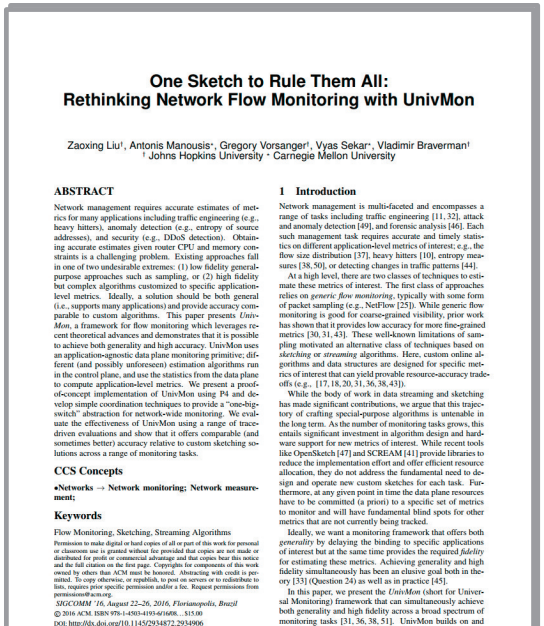
Flow-based measurements such as NetFlow [2] and sFlow [42] provide generic support for different measurement tasks, but consume too resources (e.g., CPU, memory, bandwidth) [28, 18, 19]. For example, to identify the big flows whose byte volumes are above a threshold (i.e., heavy hitter detection which is important for traffic engineering in data centers [6]), NetFlow collects flow-level counts for *sampled* packets in the data plane. A high sampling rate would lead to too many counters, while a lower sampling rate may miss flows. While there are many NetFlow improvements for specific measurement tasks (e.g., [48, 19]), a different measurement task may need to focus on small flows (e.g., anomaly detection) and thus requiring another way of changing NetFlow. Instead, *we should provide more customized and dynamic measurement data collection defined by the software written by operators based on the measurement requirements; and provide guarantees on the measurement accuracy.*

As an alternative, many *sketch*-based streaming algorithms have been proposed in the theoretical research community [7, 12, 46, 8, 20, 47], which provide efficient measurement support for individual management tasks. However, these algorithms are not deployed in practice because of their lack of generality: Each of these algorithms answers just one question or produces just one statistic (e.g., the unique number of destinations), so it is too expensive for vendors to build new hardware to support each function. For example, the Space-Saving heavy hitter detection algorithm [8] maintains a hash table of items and counts, and requires customized operations such as keeping a pointer to the item with minimum counts and replacing the minimum-count entry with a

UnivMon

SIGCOMM ‘16

[source]



One Sketch to Rule Them All:
Rethinking Network Flow Monitoring with UnivMon

Zaoxing Liu†, Antonis Manousis⋆, Gregory Vorsanger†, Vyas Sekar⋆, Vladimir Braverman†
† Johns Hopkins University ⋆ Carnegie Mellon University

ABSTRACT

Network management requires accurate estimates of metrics for many applications including traffic engineering (e.g., heavy hitters), anomaly detection (e.g., entropy of source addresses), and security (e.g., DDoS detection). Obtaining accurate estimates given router CPU and memory constraints is a challenging problem. Existing approaches fall in one of two undesirable extremes: (1) low fidelity general-purpose approaches such as sampling, or (2) high fidelity but complex algorithms customized to specific application-level metrics. Ideally, a solution should be both general (i.e., supports many applications) and provide accuracy comparable to custom algorithms. This paper presents *UnivMon*, a framework for flow monitoring which leverages recent theoretical advances and demonstrates that it is possible to achieve both generality and high accuracy. UnivMon uses an application-agnostic data plane monitoring primitive; different (and possibly unforeseen) estimation algorithms run in the control plane, and use the statistics from the data plane to compute application-level metrics. We present a proof-of-concept implementation of UnivMon using P4 and develop simple coordination techniques to provide a "one-big-switch" abstraction for network-wide monitoring. We evaluate the effectiveness of UnivMon using a range of trace-driven evaluations and show that it offers comparable (and sometimes better) accuracy relative to custom sketching solutions across a range of monitoring tasks.

CCS Concepts

•**Networks** → **Network monitoring; Network measurement;**

Keywords

Flow Monitoring, Sketching, Streaming Algorithms

SIGCOMM '16, August 22–26, 2016, Florianopolis, Brazil

DOI: http://dx.doi.org/10.1145/2934872.2934906

1 Introduction

Network management is multi-faceted and encompasses a range of tasks including traffic engineering [11, 32], attack and anomaly detection [49], and forensic analysis [46]. Each such management task requires accurate and timely statistics on different application-level metrics of interest; e.g., the flow size distribution [37], heavy hitters [10], entropy measures [38, 50], or detecting changes in traffic patterns [44].

At a high level, there are two classes of techniques to estimate these metrics of interest. The first class of approaches relies on *generic flow monitoring*, typically with some form of packet sampling (e.g., NetFlow [25]). While generic flow monitoring is good for coarse-grained visibility, prior work has shown that it provides low accuracy for more fine-grained metrics [30, 31, 43]. These well-known limitations of sampling motivated an alternative class of techniques based on *sketching* or *streaming* algorithms. Here, custom online algorithms and data structures are designed for specific metrics of interest that can yield provable resource-accuracy trade-offs (e.g., [17, 18, 20, 31, 36, 38, 43]).

While the body of work in data streaming and sketching has made significant contributions, we argue that this trajectory of crafting special-purpose algorithms is untenable in the long term. As the number of monitoring tasks grows, this entails significant investment in algorithm design and hardware support for new metrics of interest. While recent tools like OpenSketch [47] and SCREAM [41] provide libraries to reduce the implementation effort and offer efficient resource allocation, they do not address the fundamental need to design and operate new custom sketches for each task. Furthermore, at any given point in time the data plane resources have to be committed (a priori) to a specific set of metrics to monitor and will have fundamental blind spots for other metrics that are not currently being tracked.

Ideally, we want a monitoring framework that offers both *generality* by delaying the binding to specific applications of interest but at the same time provides the required *fidelity* for estimating these metrics. Achieving generality and high fidelity simultaneously has been an elusive goal both in theory [33] (Question 24) as well as in practice [45].

In this paper, we present the *UnivMon* (short for Universal Monitoring) framework that can simultaneously achieve both generality and high fidelity across a broad spectrum of monitoring tasks [31, 36, 38, 51]. UnivMon builds on and

SketchLearn

SIGCOMM ‘18

[source]

SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference

Qun Huang†, Patrick P. C. Lee‡, and Yungang Bao†
†State Key Lab of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
‡Department of Computer Science and Engineering, The Chinese University of Hong Kong

ABSTRACT

Network measurement is challenged to fulfill stringent resource requirements in the face of massive network traffic. While approximate measurement can trade accuracy for resource savings, it demands intensive manual efforts to configure the right resource-accuracy trade-offs in real deployment. Such user burdens are caused by how existing approximate measurement approaches inherently deal with resource conflicts when tracking massive network traffic with limited resources. In particular, they tightly couple resource configurations with accuracy parameters, so as to provision sufficient resources to bound the measurement errors. We design SketchLearn, a novel sketch-based measurement framework that resolves resource conflicts by learning their statistical properties to eliminate conflicting traffic components. We prototype SketchLearn on OpenVSwitch and P4, and our testbed experiments and stress-test simulation show that SketchLearn accurately and automatically monitors various traffic statistics and effectively supports network-wide measurement with limited resources.

CCS CONCEPTS

• **Networks** → **Network measurement**;

KEYWORDS

Sketch; Network measurement

ACM Reference Format:
Qun Huang, Patrick P. C. Lee, and Yungang Bao. 2018. SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference. In *SIGCOMM '18: ACM SIGCOMM*

SIGCOMM '18, August 20–25, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5567-4/18/08…$15.00
https://doi.org/10.1145/3230543.3230559

2018 Conference, August 20–25, 2018, Budapest, Hungary. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3230543.3230559

1 INTRODUCTION

Network measurement is indispensable to modern network management in clouds and data centers. Administrators measure a variety of traffic statistics, such as per-flow frequency, to infer the key behaviors or any unexpected patterns in operational networks. They use the measured traffic statistics to form the basis of management operations such as traffic engineering, performance diagnosis, and intrusion prevention. Unfortunately, measuring traffic statistics is non-trivial in the face of massive network traffic and large-scale network deployment. Error-free measurement requires per-flow tracking [15], yet today's data center networks can have thousands of concurrent flows in a very small period from 50ms [2] down to even 5ms [56]. This would require tremendous resources for performing per-flow tracking.

In view of the resource constraints, many approaches in the literature leverage approximation techniques to trade between resource usage and measurement accuracy. Examples include sampling [9, 37, 64], top-k counting [5, 43, 44, 46], and sketch-based approaches [18, 33, 40, 42, 58], which we collectively refer to as *approximate measurement* approaches. Their idea is to construct compact sub-linear data structures to record traffic statistics, backed by theoretical guarantees on how to achieve accurate measurement with limited resources. Approximate measurement has formed building blocks in many state-of-the-art network-wide measurement systems (e.g., [32, 48, 55, 60, 62, 67]), and is also adopted in production data centers [31, 68].

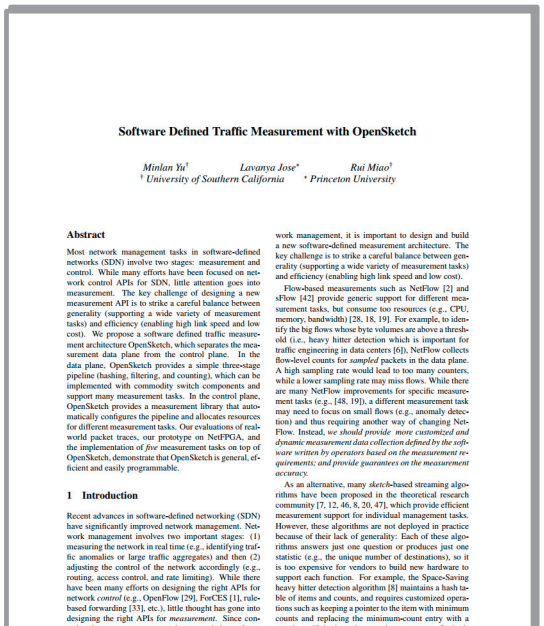
Although theoretically sound, existing approximate measurement approaches are inconvenient for use. In such approaches, massive network traffic competes for the limited resources, thereby introducing measurement errors due to *resource conflicts* (e.g., multiple flows are mapped to the same counter in sketch-based measurement). To mitigate errors, sufficient resources must be provisioned in approximate measurement based on its theoretical guarantees. Thus, *there exists a tight binding between resource configurations and accuracy parameters.* Such tight binding leads to several practical limitations (see §2.2 for details): (i) administrators need

Sketches **are the new black**

OpenSketch

NSDI ‘13

[source]



Software Defined Traffic Measurement with OpenSketch

Minlan Yu† Lavanya Jose* Rui Miao†
† University of Southern California * Princeton University

Abstract

Most network management tasks in software-defined networks (SDN) involve two stages: measurement and control. While many efforts have been focused on network control APIs for SDN, little attention goes into measurement. The key challenge of designing a new measurement API is to strike a careful balance between generality (supporting a wide variety of measurement tasks) and efficiency (enabling high link speed and low cost). We propose a software defined traffic measurement architecture OpenSketch, which separates the measurement data plane from the control plane. In the data plane, OpenSketch provides a simple three-stage pipeline (hashing, filtering, and counting), which can be implemented with commodity switch components and support many measurement tasks. In the control plane, OpenSketch provides a measurement library that automatically configures the pipeline and allocates resources for different measurement tasks. Our evaluations of real-world packet traces, our prototype on NetFPGA, and the implementation of *five* measurement tasks on top of OpenSketch, demonstrate that OpenSketch is general, efficient and easily programmable.

1 Introduction

Recent advances in software-defined networking (SDN) have significantly improved network management. Network management involves two important stages: (1) measuring the network in real time (e.g., identifying traffic anomalies or large traffic aggregates) and then (2) adjusting the control of the network accordingly (e.g., routing, access control, and rate limiting). While there have been many efforts on designing the right APIs for network *control* (e.g., OpenFlow [29], ForCES [1], rule-based forwarding [33], etc.), little thought has gone into designing the right APIs for *measurement*. Since network management, it is important to design and build a new software-defined measurement architecture. The key challenge is to strike a careful balance between generality (supporting a wide variety of measurement tasks) and efficiency (enabling high link speed and low cost).

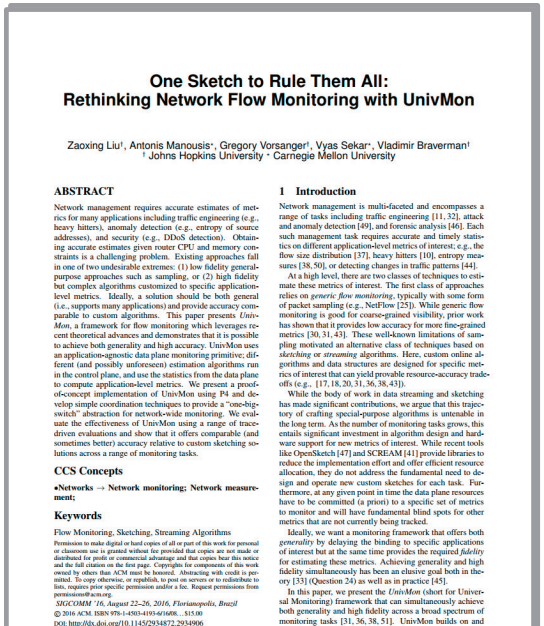
Flow-based measurements such as NetFlow [2] and sFlow [42] provide generic support for different measurement tasks, but consume too resources (e.g., CPU, memory, bandwidth) [28, 18, 19]. For example, to identify the big flows whose byte volumes are above a threshold (i.e., heavy hitter detection which is important for traffic engineering in data centers [6]), NetFlow collects flow-level counts for *sampled* packets in the data plane. A high sampling rate would lead to too many counters, while a lower sampling rate may miss flows. While there are many NetFlow improvements for specific measurement tasks (e.g., [48, 19]), a different measurement task may need to focus on small flows (e.g., anomaly detection) and thus requiring another way of changing NetFlow. Instead, *we should provide more customized and dynamic measurement data collection defined by the software written by operators based on the measurement requirements; and provide guarantees on the measurement accuracy.*

As an alternative, many *sketch*-based streaming algorithms have been proposed in the theoretical research community [7, 12, 46, 8, 20, 47], which provide efficient measurement support for individual management tasks. However, these algorithms are not deployed in practice because of their lack of generality: Each of these algorithms answers just one question or produces just one statistic (e.g., the unique number of destinations), so it is too expensive for vendors to build new hardware to support each function. For example, the Space-Saving heavy hitter detection algorithm [8] maintains a hash table of items and counts, and requires customized operations such as keeping a pointer to the item with minimum counts and replacing the minimum-count entry with a

UnivMon

SIGCOMM ‘16

[source]



One Sketch to Rule Them All:
Rethinking Network Flow Monitoring with UnivMon

Zaoxing Liu†, Antonis Manousis⋆, Gregory Vorsanger†, Vyas Sekar⋆, Vladimir Braverman†
† Johns Hopkins University ⋆ Carnegie Mellon University

ABSTRACT

Network management requires accurate estimates of metrics for many applications including traffic engineering (e.g., heavy hitters), anomaly detection (e.g., entropy of source addresses), and security (e.g., DDoS detection). Obtaining accurate estimates given router CPU and memory constraints is a challenging problem. Existing approaches fall in one of two undesirable extremes: (1) low fidelity general-purpose approaches such as sampling, or (2) high fidelity but complex algorithms customized to specific application-level metrics. Ideally, a solution should be both general (i.e., supports many applications) and provide accuracy comparable to custom algorithms. This paper presents *UnivMon*, a framework for flow monitoring which leverages recent theoretical advances and demonstrates that it is possible to achieve both generality and high accuracy. UnivMon uses an application-agnostic data plane monitoring primitive; different (and possibly unforeseen) estimation algorithms run in the control plane, and use the statistics from the data plane to compute application-level metrics. We present a proof-of-concept implementation of UnivMon using P4 and develop simple coordination techniques to provide a "one-big-switch" abstraction for network-wide monitoring. We evaluate the effectiveness of UnivMon using a range of trace-driven evaluations and show that it offers comparable (and sometimes better) accuracy relative to custom sketching solutions across a range of monitoring tasks.

CCS Concepts

•**Networks** → **Network monitoring; Network measurement;**

Keywords

Flow Monitoring, Sketching, Streaming Algorithms

SIGCOMM '16, August 22–26, 2016, Florianopolis, Brazil

DOI: http://dx.doi.org/10.1145/2934872.2934906

1 Introduction

Network management is multi-faceted and encompasses a range of tasks including traffic engineering [11, 32], attack and anomaly detection [49], and forensic analysis [46]. Each such management task requires accurate and timely statistics on different application-level metrics of interest; e.g., the flow size distribution [37], heavy hitters [10], entropy measures [38, 50], or detecting changes in traffic patterns [44].

At a high level, there are two classes of techniques to estimate these metrics of interest. The first class of approaches relies on *generic flow monitoring*, typically with some form of packet sampling (e.g., NetFlow [25]). While generic flow monitoring is good for coarse-grained visibility, prior work has shown that it provides low accuracy for more fine-grained metrics [30, 31, 43]. These well-known limitations of sampling motivated an alternative class of techniques based on *sketching* or *streaming* algorithms. Here, custom online algorithms and data structures are designed for specific metrics of interest that can yield provable resource-accuracy trade-offs (e.g., [17, 18, 20, 31, 36, 38, 43]).

While the body of work in data streaming and sketching has made significant contributions, we argue that this trajectory of crafting special-purpose algorithms is untenable in the long term. As the number of monitoring tasks grows, this entails significant investment in algorithm design and hardware support for new metrics of interest. While recent tools like OpenSketch [47] and SCREAM [41] provide libraries to reduce the implementation effort and offer efficient resource allocation, they do not address the fundamental need to design and operate new custom sketches for each task. Furthermore, at any given point in time the data plane resources have to be committed (a priori) to a specific set of metrics to monitor and will have fundamental blind spots for other metrics that are not currently being tracked.

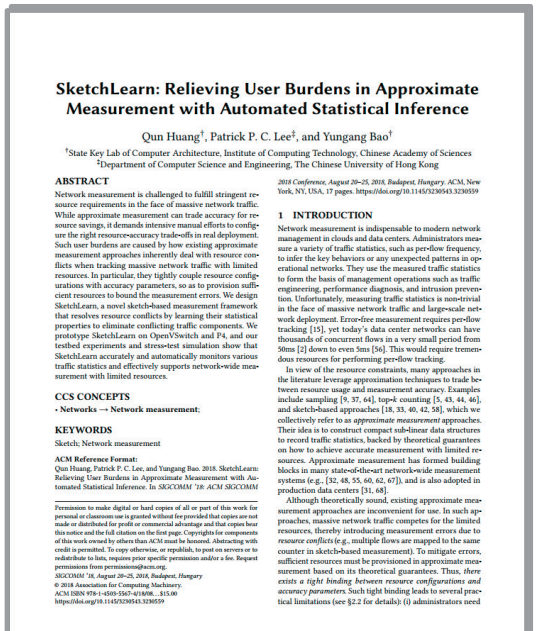
Ideally, we want a monitoring framework that offers both *generality* by delaying the binding to specific applications of interest but at the same time provides the required *fidelity* for estimating these metrics. Achieving generality and high fidelity simultaneously has been an elusive goal both in theory [33] (Question 24) as well as in practice [45].

In this paper, we present the *UnivMon* (short for Universal Monitoring) framework that can simultaneously achieve both generality and high fidelity across a broad spectrum of monitoring tasks [31, 36, 38, 51]. UnivMon builds on and

SketchLearn

SIGCOMM ‘18

[source]



SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference

Qun Huang†, Patrick P. C. Lee‡, and Yungang Bao†
†State Key Lab of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
‡Department of Computer Science and Engineering, The Chinese University of Hong Kong

ABSTRACT

Network measurement is challenged to fulfill stringent resource requirements in the face of massive network traffic. While approximate measurement can trade accuracy for resource savings, it demands intensive manual efforts to configure the right resource-accuracy trade-offs in real deployment. Such user burdens are caused by how existing approximate measurement approaches inherently deal with resource conflicts when tracking massive network traffic with limited resources. In particular, they tightly couple resource configurations with accuracy parameters, so as to provision sufficient resources to bound the measurement errors. We design SketchLearn, a novel sketch-based measurement framework that resolves resource conflicts by learning their statistical properties to eliminate conflicting traffic components. We prototype SketchLearn on OpenVSwitch and P4, and our testbed experiments and stress-test simulation show that SketchLearn accurately and automatically monitors various traffic statistics and effectively supports network-wide measurement with limited resources.

CCS CONCEPTS

• **Networks** → **Network measurement**;

KEYWORDS

Sketch; Network measurement

ACM Reference Format:
Qun Huang, Patrick P. C. Lee, and Yungang Bao. 2018. SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference. In *SIGCOMM '18: ACM SIGCOMM 2018 Conference, August 20–25, 2018, Budapest, Hungary.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3230543.3230559

SIGCOMM '18, August 20–25, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5567-4/18/08…$15.00
https://doi.org/10.1145/3230543.3230559

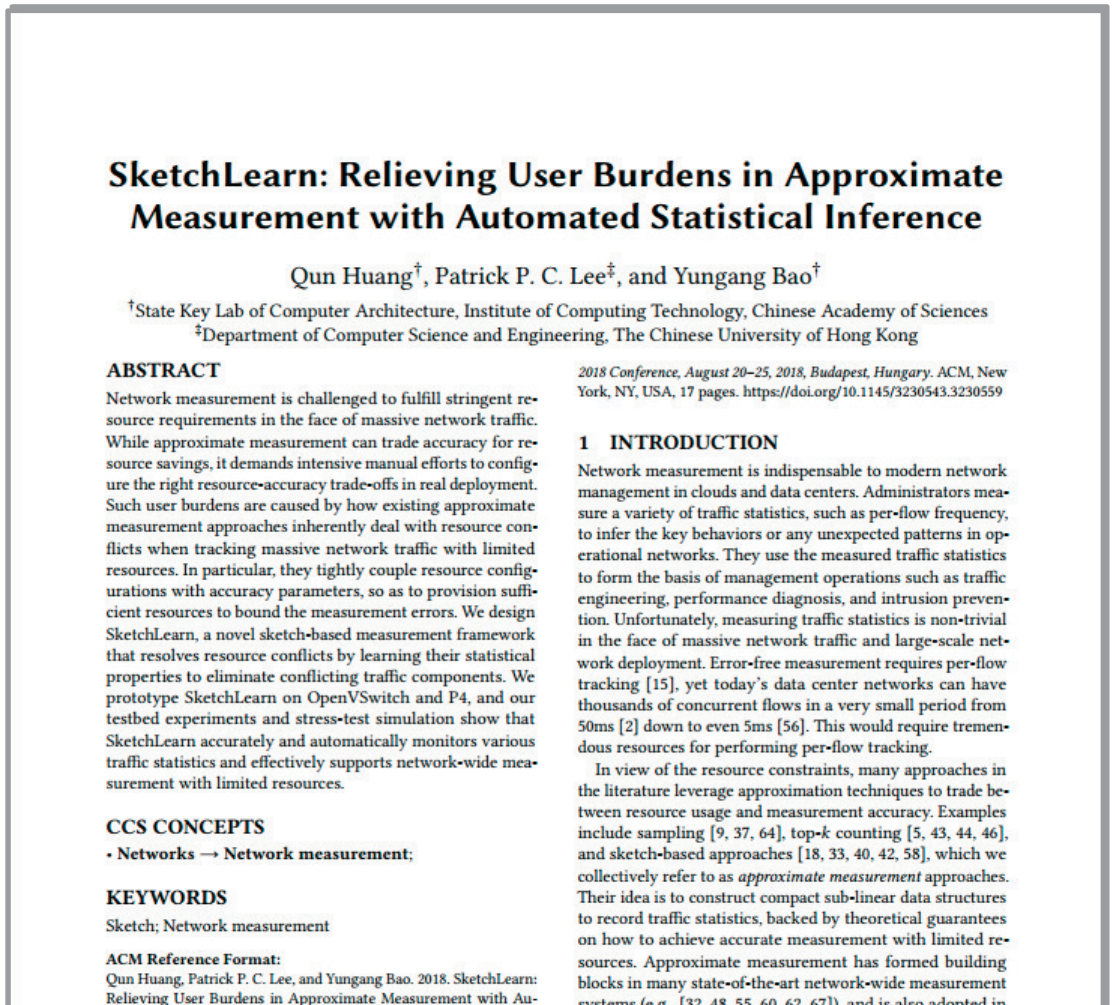
1 INTRODUCTION

Network measurement is indispensable to modern network management in clouds and data centers. Administrators measure a variety of traffic statistics, such as per-flow frequency, to infer the key behaviors or any unexpected patterns in operational networks. They use the measured traffic statistics to form the basis of management operations such as traffic engineering, performance diagnosis, and intrusion prevention. Unfortunately, measuring traffic statistics is non-trivial in the face of massive network traffic and large-scale network deployment. Error-free measurement requires per-flow tracking [15], yet today's data center networks can have thousands of concurrent flows in a very small period from 50ms [2] down to even 5ms [56]. This would require tremendous resources for performing per-flow tracking.

In view of the resource constraints, many approaches in the literature leverage approximation techniques to trade between resource usage and measurement accuracy. Examples include sampling [9, 37, 64], top-k counting [5, 43, 44, 46], and sketch-based approaches [18, 33, 40, 42, 58], which we collectively refer to as *approximate measurement* approaches. Their idea is to construct compact sub-linear data structures to record traffic statistics, backed by theoretical guarantees on how to achieve accurate measurement with limited resources. Approximate measurement has formed building blocks in many state-of-the-art network-wide measurement systems (e.g., [32, 48, 55, 60, 62, 67]), and is also adopted in production data centers [31, 68].

Although theoretically sound, existing approximate measurement approaches are inconvenient for use. In such approaches, massive network traffic competes for the limited resources, thereby introducing measurement errors due to *resource conflicts* (e.g., multiple flows are mapped to the same counter in sketch-based measurement). To mitigate errors, sufficient resources must be provisioned in approximate measurement based on its theoretical guarantees. Thus, *there exists a tight binding between resource configurations and accuracy parameters.* Such tight binding leads to several practical limitations (see §2.2 for details): (i) administrators need

SketchLearn combines multiple sketches with elaborate **post-processing for flexibility**



SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference

Qun Huang[†], Patrick P. C. Lee[‡], and Yungang Bao[†]

[†]State Key Lab of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
[‡]Department of Computer Science and Engineering, The Chinese University of Hong Kong

ABSTRACT

Network measurement is challenged to fulfill stringent resource requirements in the face of massive network traffic. While approximate measurement can trade accuracy for resource savings, it demands intensive manual efforts to configure the right resource-accuracy trade-offs in real deployment. Such user burdens are caused by how existing approximate measurement approaches inherently deal with resource conflicts when tracking massive network traffic with limited resources. In particular, they tightly couple resource configurations with accuracy parameters, so as to provision sufficient resources to bound the measurement errors. We design SketchLearn, a novel sketch-based measurement framework that resolves resource conflicts by learning their statistical properties to eliminate conflicting traffic components. We prototype SketchLearn on OpenVSwitch and P4, and our testbed experiments and stress-test simulation show that SketchLearn accurately and automatically monitors various traffic statistics and effectively supports network-wide measurement with limited resources.

CCS CONCEPTS

• **Networks** → **Network measurement**;

KEYWORDS

Sketch; Network measurement

ACM Reference Format:
Qun Huang, Patrick P. C. Lee, and Yungang Bao. 2018. SketchLearn:

2018 Conference, August 20–25, 2018, Budapest, Hungary. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3230543.3230559

1 INTRODUCTION

Network measurement is indispensable to modern network management in clouds and data centers. Administrators measure a variety of traffic statistics, such as per-flow frequency, to infer the key behaviors or any unexpected patterns in operational networks. They use the measured traffic statistics to form the basis of management operations such as traffic engineering, performance diagnosis, and intrusion prevention. Unfortunately, measuring traffic statistics is non-trivial in the face of massive network traffic and large-scale network deployment. Error-free measurement requires per-flow tracking [15], yet today's data center networks can have thousands of concurrent flows in a very small period from 50ms [2] down to even 5ms [56]. This would require tremendous resources for performing per-flow tracking.

In view of the resource constraints, many approaches in the literature leverage approximation techniques to trade between resource usage and measurement accuracy. Examples include sampling [9, 37, 64], top-k counting [5, 43, 44, 46], and sketch-based approaches [18, 33, 40, 42, 58], which we collectively refer to as *approximate measurement* approaches. Their idea is to construct compact sub-linear data structures to record traffic statistics, backed by theoretical guarantees on how to achieve accurate measurement with limited resources. Approximate measurement has formed building blocks in many state-of-the-art network-wide measurement

Today we'll talk about: important questions,

how 'sketches' answer them,

limitations of 'sketches',

and my master thesis :)

Sketches **compute statistical summaries**, favoring elements with **high frequency**.

$$Pr\left[\hat{x}_i - x_i \geq \varepsilon \left\|\boldsymbol{x}\right\|_1\right] \leq \delta$$

estimation error

relative to sum of all elements

Sketches **compute statistical summaries**, favoring elements with **high frequency**.

$$\textit{Let} \quad \varepsilon = 0.01, \quad \|\boldsymbol{x}\|_1 = 10000 \quad \left(\Rightarrow \varepsilon \cdot \|\boldsymbol{x}\|_1 = 100\right)$$

$$\textit{Assume two flows} \quad x_a, \quad x_b,$$

$$\textit{with} \quad \|x_a\|_1 = 1000, \quad \|x_b\|_1 = 50$$

high frequency

low frequency

Sketches **compute statistical summaries**, favoring elements with **high frequency**.

$$Let \quad \varepsilon = 0.01, \quad \|\boldsymbol{x}\|_1 = 10000 \quad \left(\Rightarrow \varepsilon \cdot \|\boldsymbol{x}\|_1 = 100\right)$$

$$Assume\ two\ flows \quad x_a\,, \quad x_b\,,$$

$$with \quad \|x_a\|_1 = 1000, \quad \|x_b\|_1 = 50$$

Error relative to **stream size**: 1%

Sketches **compute statistical summaries**, favoring elements with **high frequency**.

$$Let \quad \varepsilon = 0.01, \quad \|\boldsymbol{x}\|_1 = 10000 \quad \left(\Rightarrow \varepsilon \cdot \|\boldsymbol{x}\|_1 = 100\right)$$

$$Assume\ two\ flows \quad x_a, \quad x_b,$$

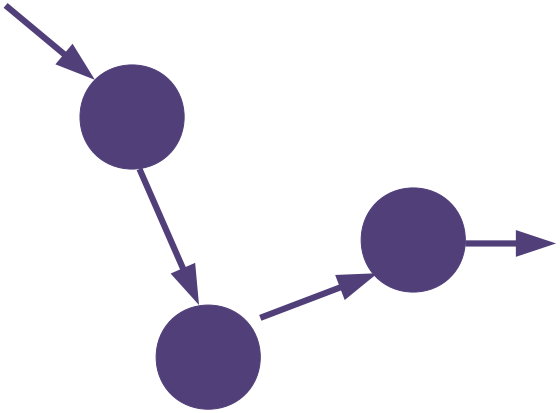
$$with \quad \|x_a\|_1 = 1000, \quad \|x_b\|_1 = 50$$

Error relative to **stream size**: 1%

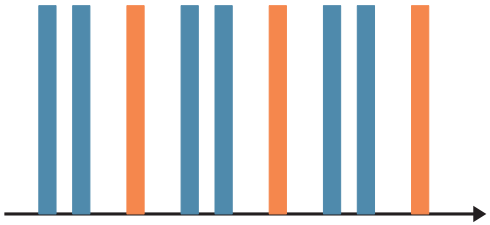
flow size: x_a: 10%, x_b: **200%**

Other Problems a Sketch **can’t handle**

causality



patterns



rare things

Regardless of their limitations, sketches provide ***trade-offs between resources and error****, and* ***provable guarantees*** *to rely on.*

Today we'll talk about: important questions,

how 'sketches' answer them,

limitations of 'sketches',

and my master thesis :)

Advanced Topics in Communication Networks

Programming Network Data Planes

Alexander Dietmüller

nsg.ee.ethz.ch

ETH Zürich

Oct. 11 2018