You are looking at a stream of data.

In networking, we usually talk about **streams of packets**, but these questions apply to other domains as well, e.g. **search engines and databases**.

You are looking at a stream of data (packets).
 There are many questions you might ask:

Is a certain element (e.g. ip address) in the stream?

How frequently does an element appear?

How many distinct elements are in the stream?

What are the most frequent elements?

You are looking at a stream of data (packets).
 There are many questions you might ask:

Is a certain element (e.g. ip address) in the stream?

How frequently does an element appear?

How many distinct elements are in the stream?

What are the most frequent elements?

In P4, these questions are difficult to answer.

You are looking at a stream of data (packets).

Today, I'll show you how set membership and frequency queries can be realized in P4.

Is a certain element (e.g. ip address) in the stream?

→ Bloom filter

How frequently does an element appear?

→ CountMin Sketch, Count Sketch, ...

part 1: set membership queries with Bloom filters

Is a certain element (e.g. ip address) in the stream?

(slides by Thomas Holterbach)

There are two common strategies to implement a set

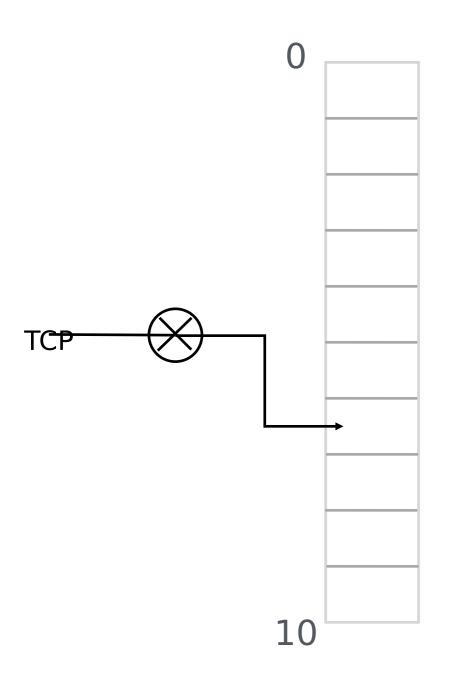
output

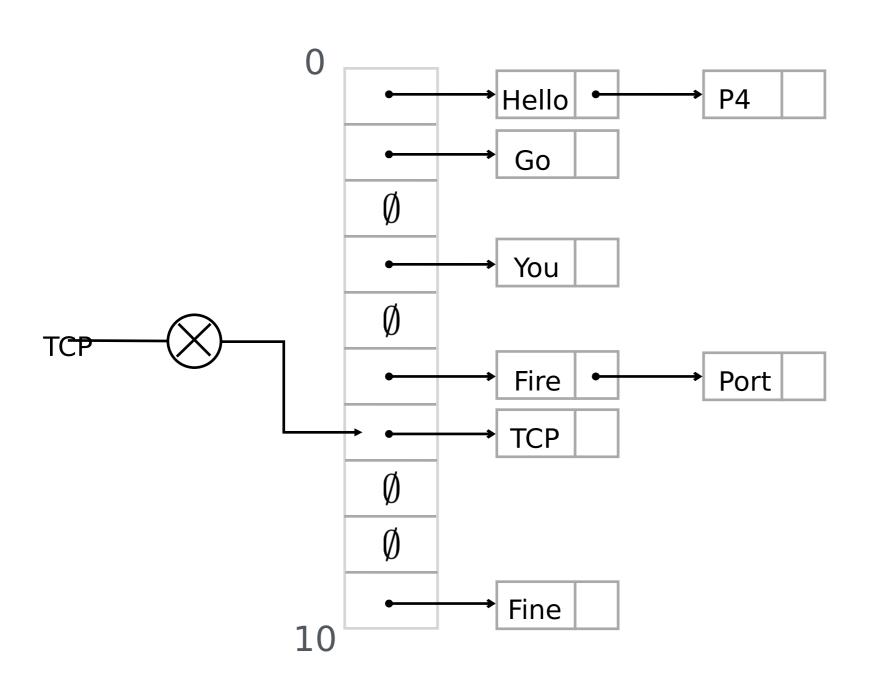
Deterministic

number of required operations

Probabilistic

Intuitive implementation of a set





N elements and M cells

list size

average N/M

worse-case N

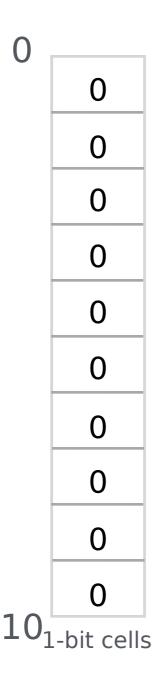
Pros: accurate and fast in the average case

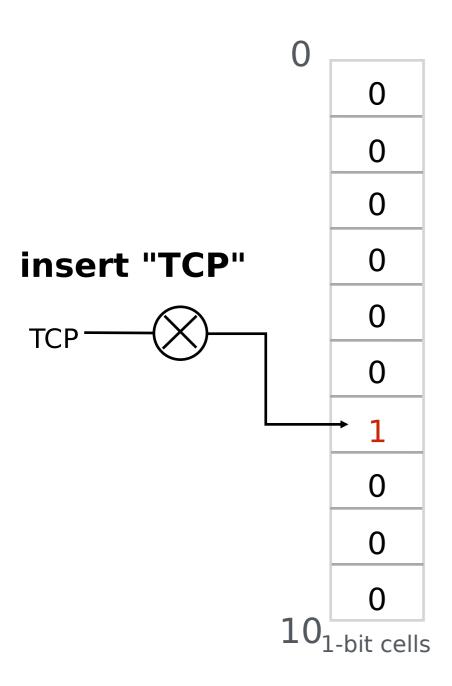
Con: only works in hardware if there is a low number of elements (e.g. < 100)

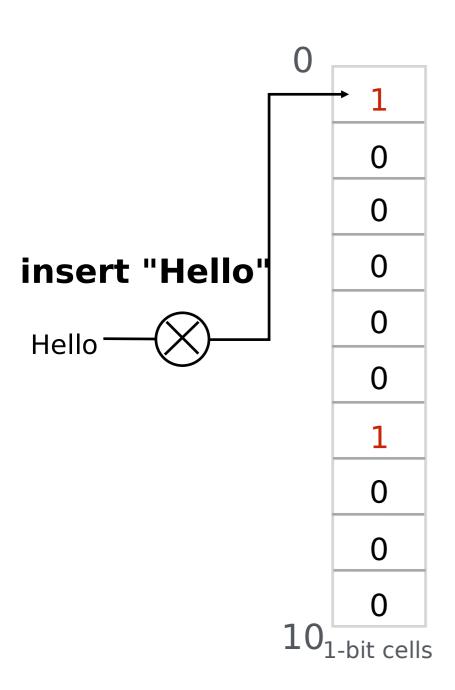
There are two common strategies to implement a set

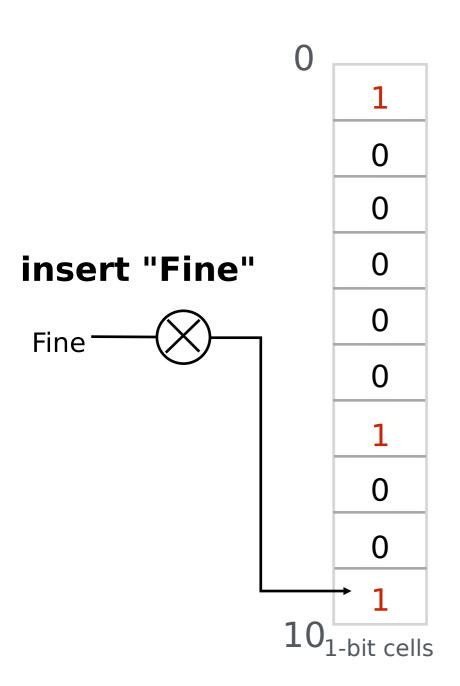
	strategy #1	strategy #2
output	Deterministic	Probabilistic
number of required operations	Probabilistic	Deterministic

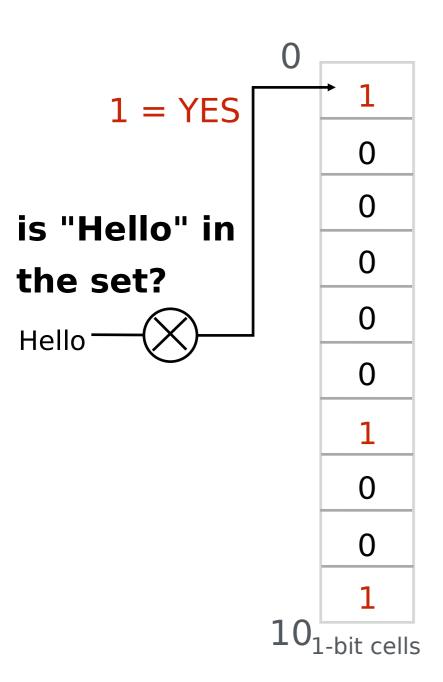
'probabilistic data structures'

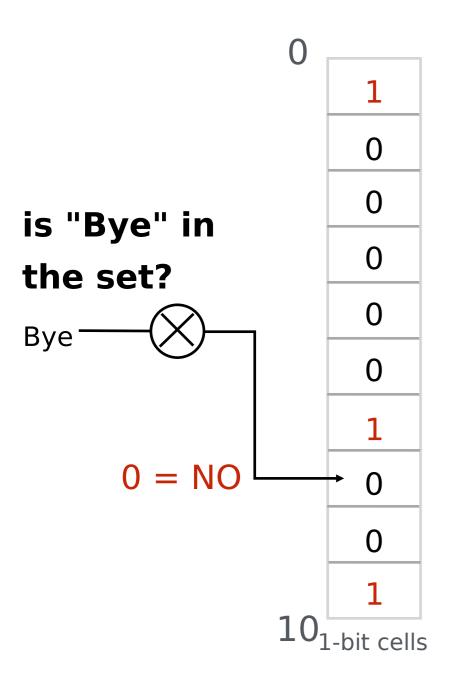


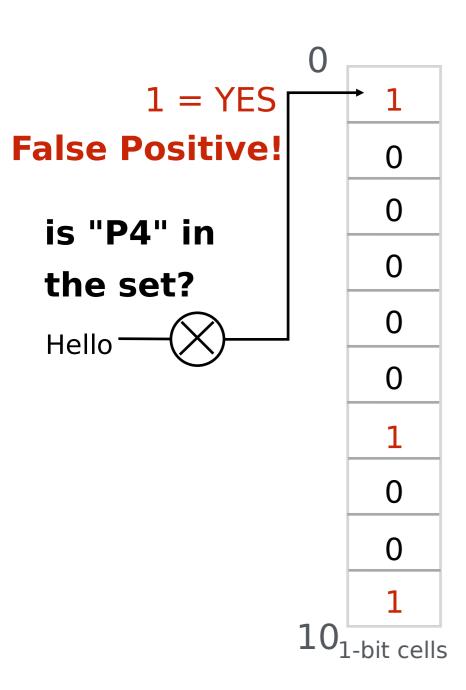












N elements and M cells

probability of an element to be mapped into a particular cell

probability of an element not to be mapped into a particular cell

probability of a cell to be 0

false positive rate (FPR)

false negative rate

$$\frac{1}{M}$$

$$1-rac{1}{M}$$

$$(1-\frac{1}{M})^N$$

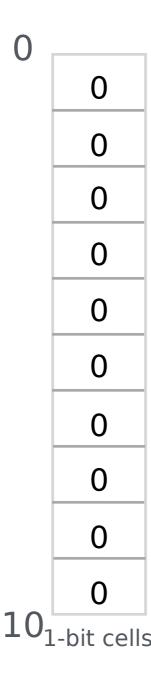
$$1 - (1 - \frac{1}{M})^N$$

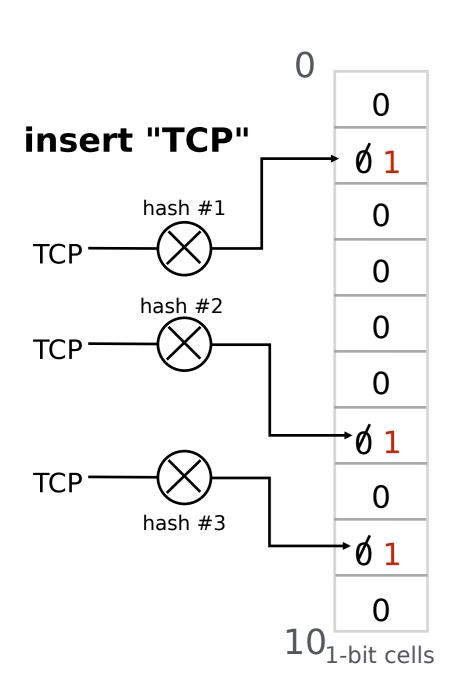
0

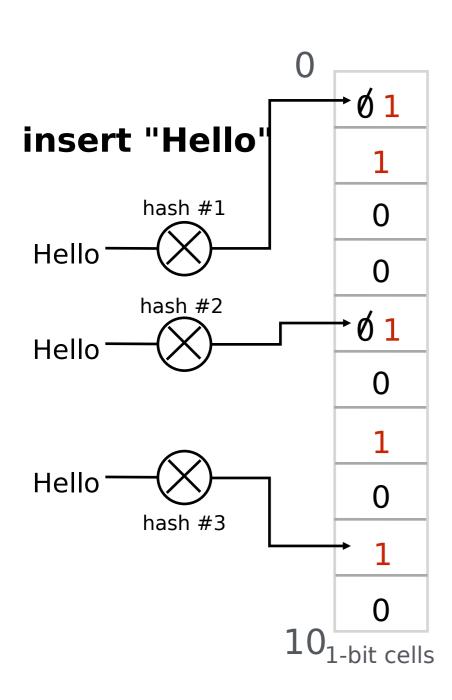
# of elements	# of cells	FPR
1000	10000	9.5%
1000	100000	1%

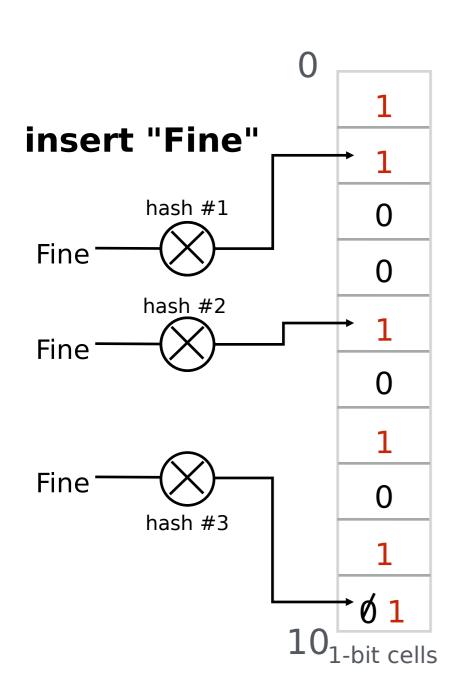
Pros: simple and only one operation per insertion or query

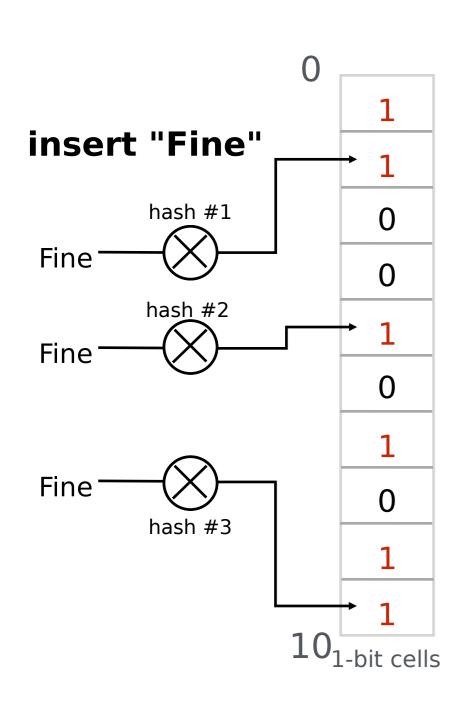
Con: roughly 100x more cells are required than the number of element we want to store for a 1% false positive rate





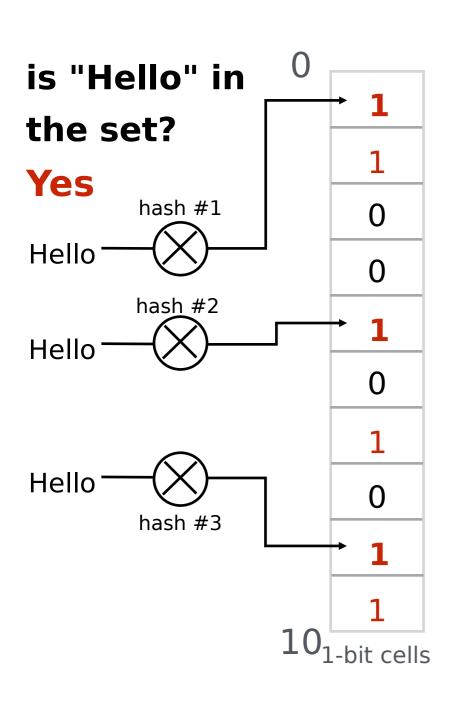






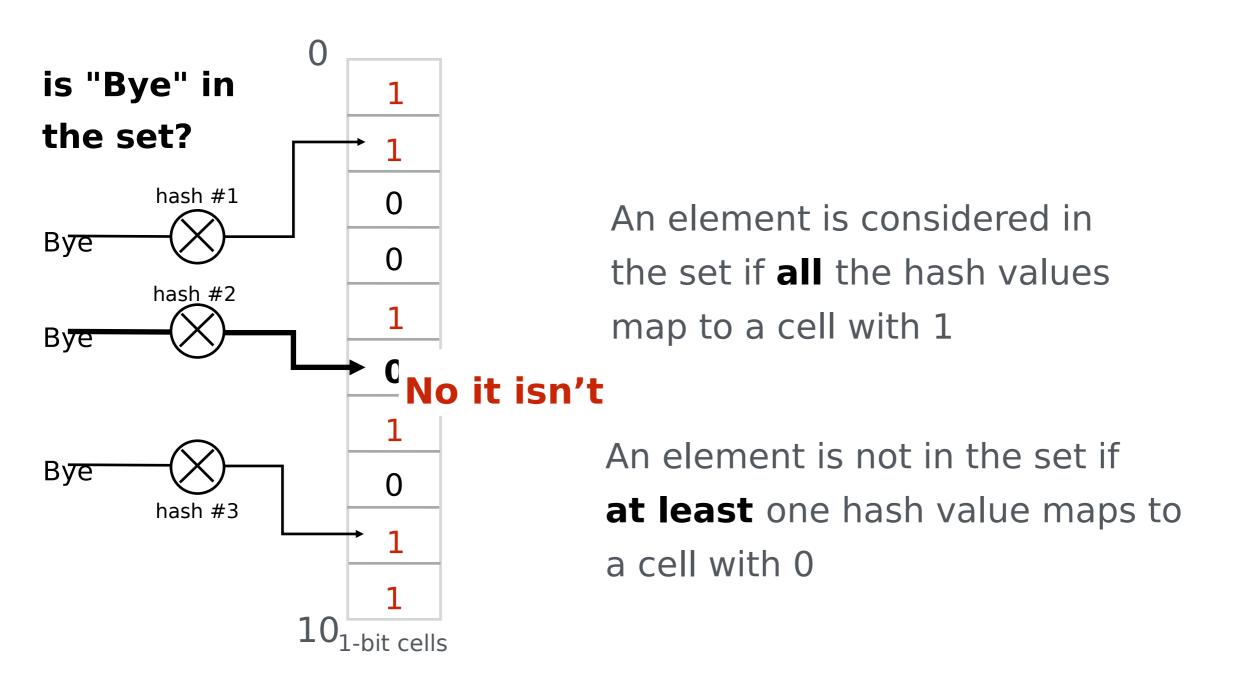
An element is considered in the set if **all** the hash values map to a cell with 1

An element is not in the set if at least one hash value maps to a cell with 0

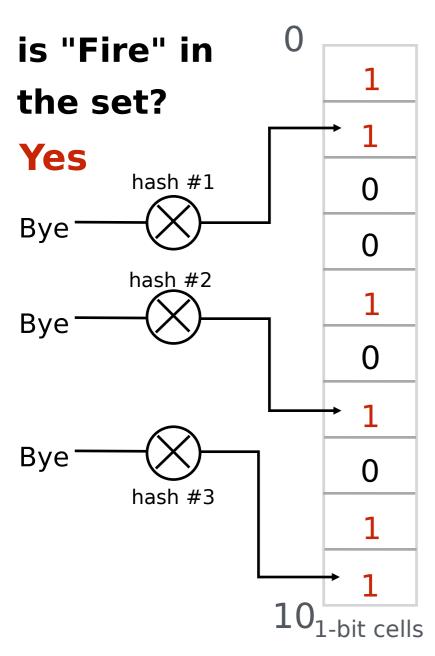


An element is considered in the set if **all** the hash values map to a cell with 1

An element is not in the set if at least one hash value maps to a cell with 0



False Positive!



An element is considered in the set if **all** the hash values map to a cell with 1

An element is not in the set if at least one hash value maps to a cell with 0

N elements, M cells and K independent hash functions

probability that one hash function returns the index of a particular cell

$$\frac{1}{M}$$

probability that one hash function does not return the index of a particular cell

$$1-rac{1}{M}$$

probability of a cell to be 0

$$(1 - \frac{1}{M})^{KN}$$

false positive rate

$$(1-(1-\frac{1}{M})^{KN})^{K}$$

false negative rate

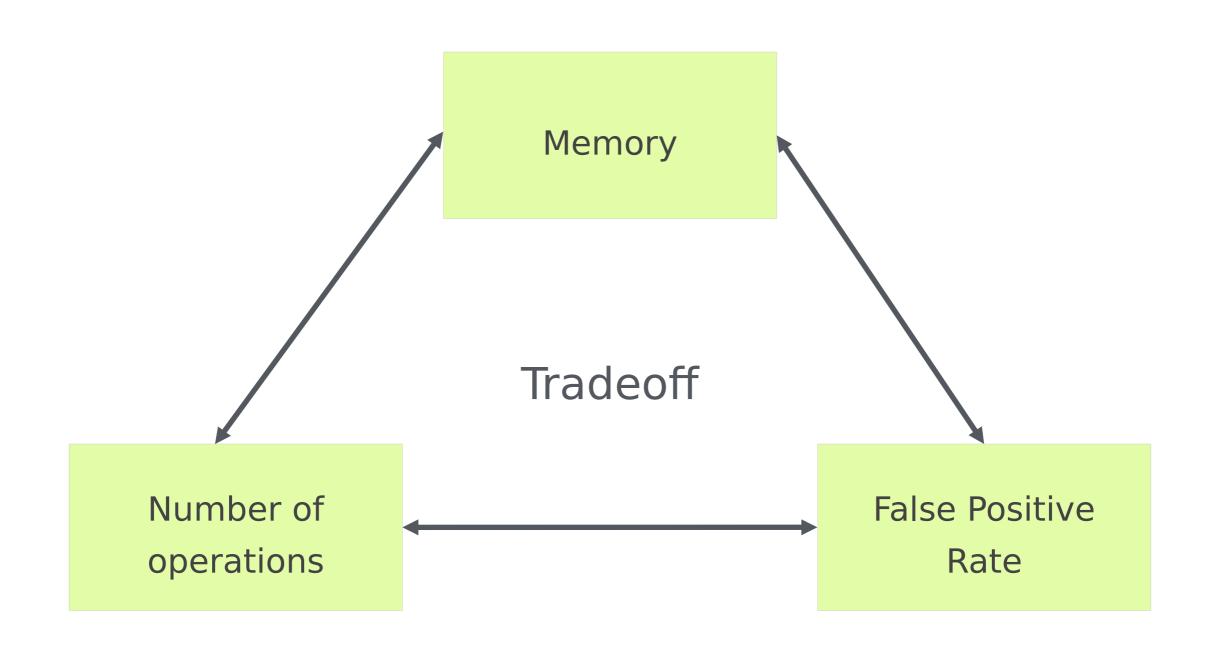
0

# of elements	# of cells	# hash functions	FPR
1000	10000	7	0.82%
1000	100000	7	\approx 0%

Pro: consumes roughly 10x less memory than the simple approach

Con: Requires slightly more operations than the simple approach (7 hashes instead of just 1)

Dimension your Bloom Filter



Dimension your Bloom Filter

N elements

M cells

K hash functions

FP false positive rate

Dimension your Bloom Filter

N elements

M cells

K hash functions

FP false positive rate

asymptotic approx.

$$FP = (1 - (1 - \frac{1}{M})^{KN})^K \approx (1 - e^{-KN/M})^K$$

with calculus you can dimension your bloom filter

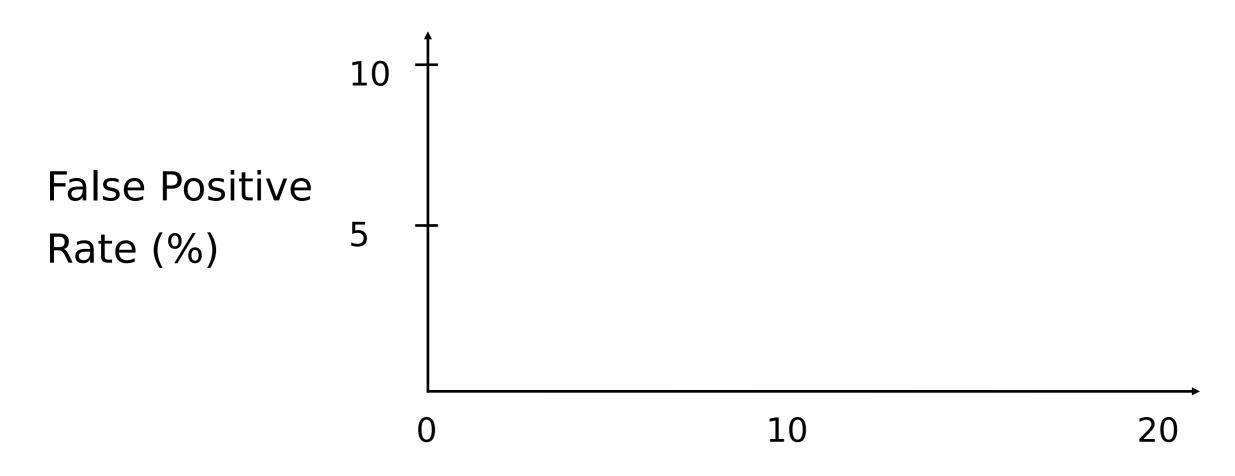
Dimension your Bloom Filter

N elements

M cells

K hash functions

FP false positive rate



K (number of hash functions)

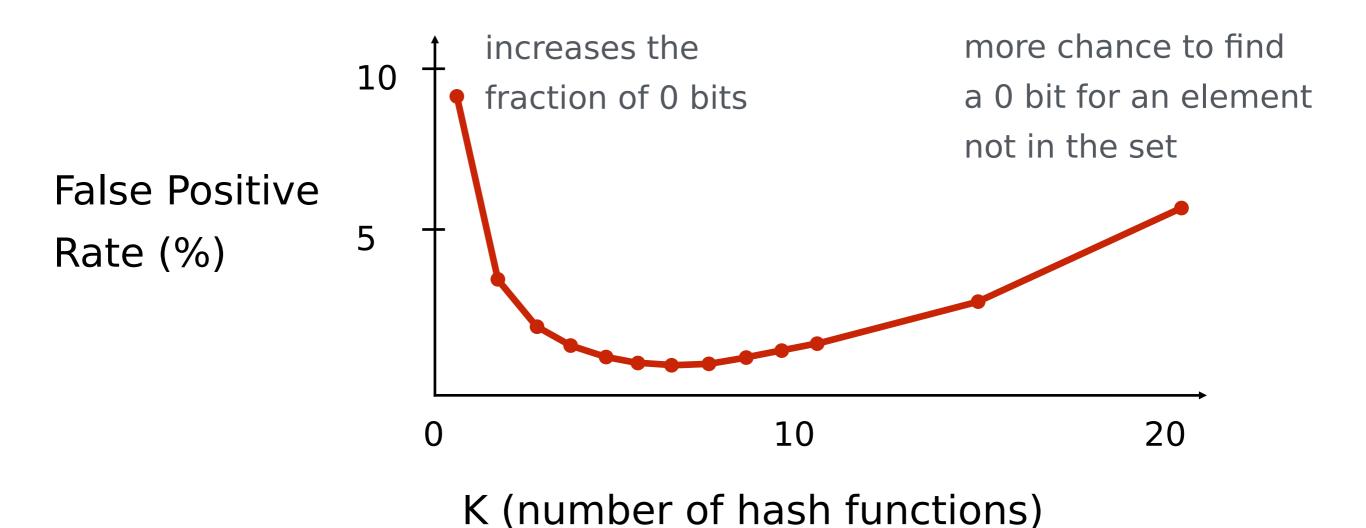
Dimension your Bloom Filter

N elements

M cells

K hash functions

FP false positive rate



Dimension your Bloom Filter

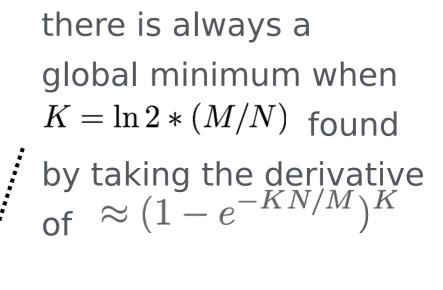
N elements

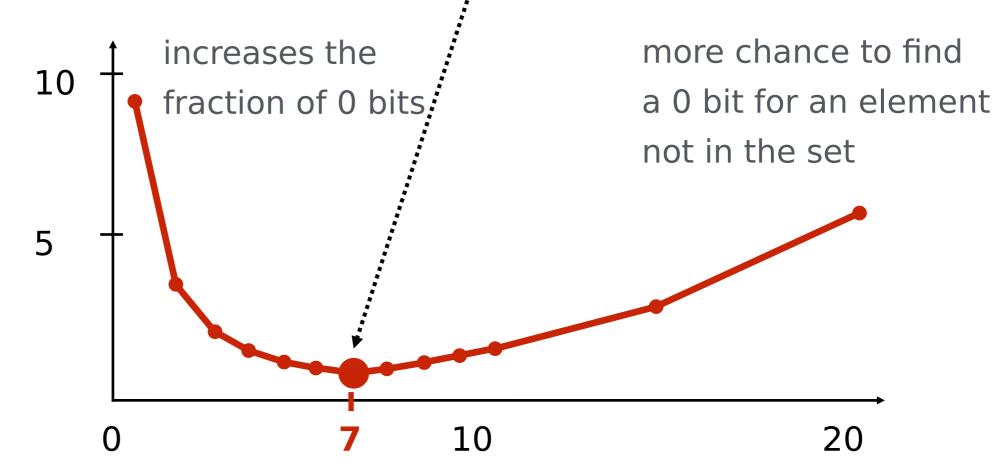
M cells

K hash functions

FP false positive rate

False Positive Rate (%)





K (number of hash functions)

Implementation of a Bloom Filter in P4₁₆

You will have to use hash functions

v1model

```
enum HashAlgorithm {
    crc32,
    crc32_custom,
    crc16,
    s,
    random,
    identity,
    csum16,
    xor16
}
```

extern void hash<0, T, D, M>(out O result, in HashAlgorithm algo, in T base, in D data, in M max);

Implementation of a Bloom Filter in P4₁₆

You will have to use hash functions, as well as registers

```
v1model
    extern register<T> {
        register(bit<32> size);
        void read(out T result, in bit<32> index);
        void write(in bit<32> index, in T value);
    }
```

```
control MyIngress(...) {
    register register<bit<1>>(NB_CELLS) bloom_filter;
```

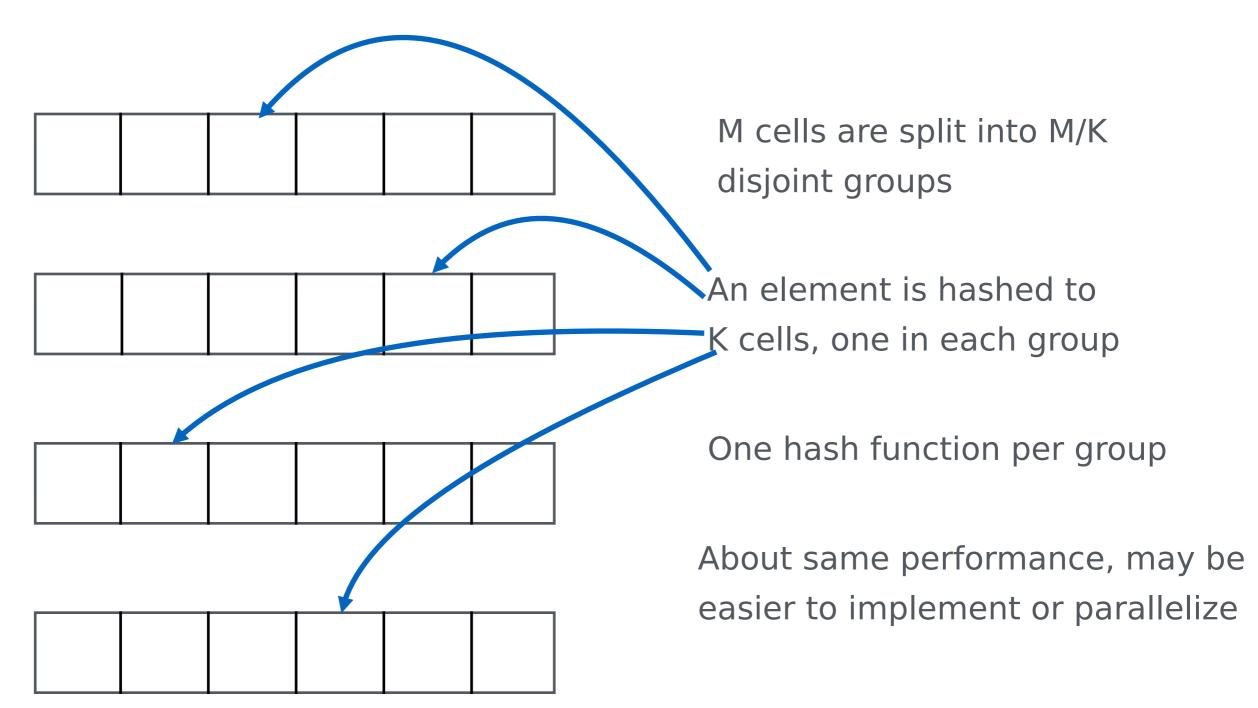
```
control MyIngress(...) {
    register register<bit<1>>(NB_CELLS) bloom_filter;
    apply {
       hash(meta.index1, HashAlgorithm.my_hash1, 0,
         {meta.dstPrefix, packet.ip.srcIP}, NB_CELLS);
       hash(meta.index2, HashAlgorithm.my_hash2, 0,
         {meta.dstPrefix, packet.ip.srcIP}, NB_CELLS);
```

```
control MyIngress(...) {
    register register<bit<1>>(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);
       if (meta.to_insert == 1) {
         bloom_filter.write(meta.index1, 1);
         bloom_filter.write(meta.index2, 1);
       if (meta.to_query == 1) {
         bloom_filter.read(meta.query1, meta.index1);
         bloom_filter.read(meta.query2, meta.index2);
         if (meta.query1 == 0 \mid | meta.query2 == 0) {
          meta.is_stored = 0;
         else {
          meta.is_stored = 1;
```

```
control MyIngress(...) {
    register register<bit<1>>(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);
       if (meta.to insert == 1) {
         bloom_filter.write(meta.index1, 1);
        bloom_filter.write(meta.index2, 1);
       if (meta.to_query == 1) {
         bloom_filter.read(meta.query1, meta.index1);
         bloom_filter.read(meta.query2, meta.index2);
        if (meta.query1 == 0 || meta.query2 == 0) {
          meta.is_stored = 0;
        else {
          meta.is_stored = 1;
```

Everything in bold red must be adapted for your program

Depending on the hardware limitations, splitting the bloom filter might be required



Because deletions are not possible, the controller may need to regularly reset the bloom filters

Resetting a bloom filter takes some time during which it is not usable

Common trick: use two bloom filters and use one when the controller resets the other one Bloom filters may be extended to allow deletions and to list the filter content.

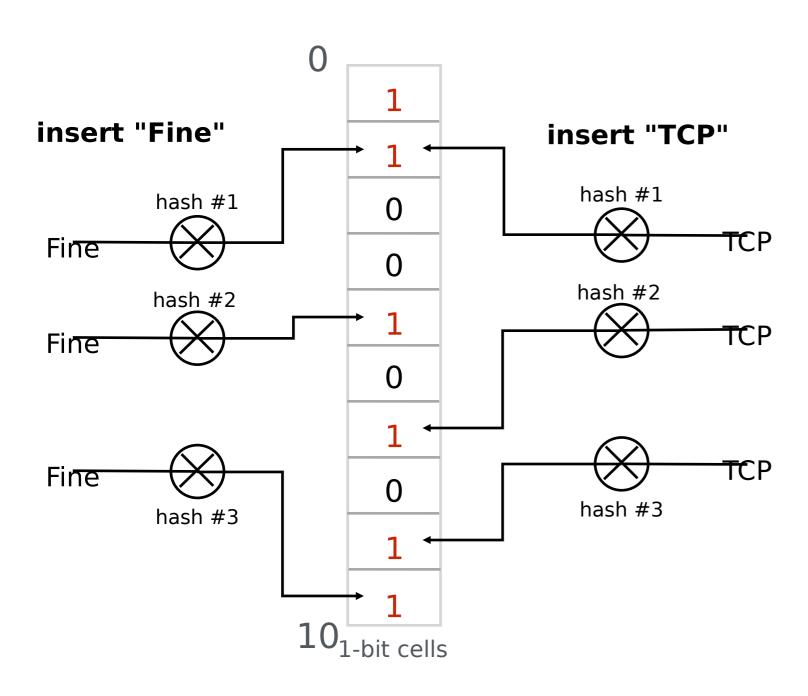
If you are curious, check out the extended slides for:

- counting Bloom filters (allow deletions)
- invertible Bloom filters (allow to list content)

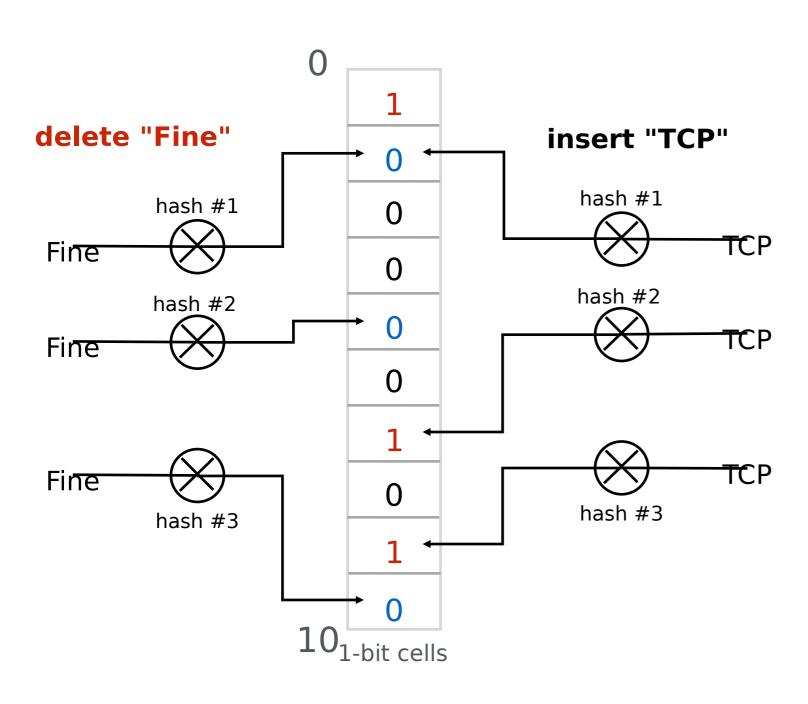
So far we have seen how to do insertions and membership queries

	strategy #1	strategy #2
output	Deterministic	Probabilistic
number of required operations	Probabilistic	Deterministic
•		
		Bloom Filters

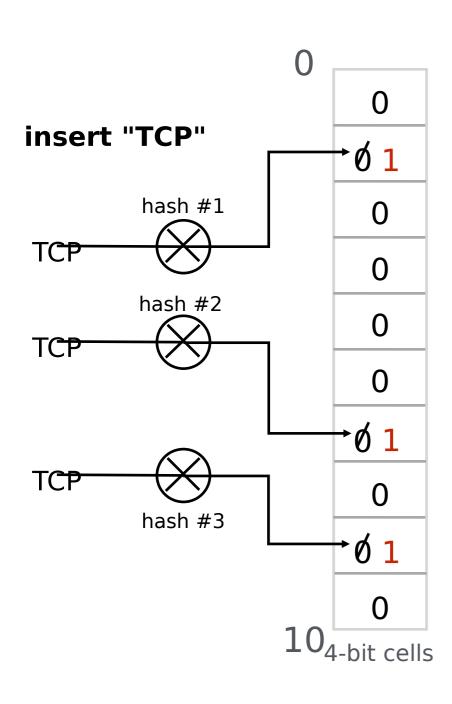
However Bloom Filters do not handle deletions



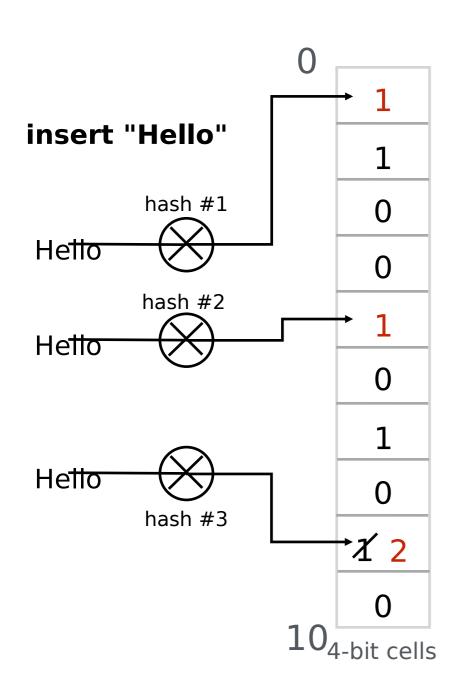
However Bloom Filters do not handle deletions



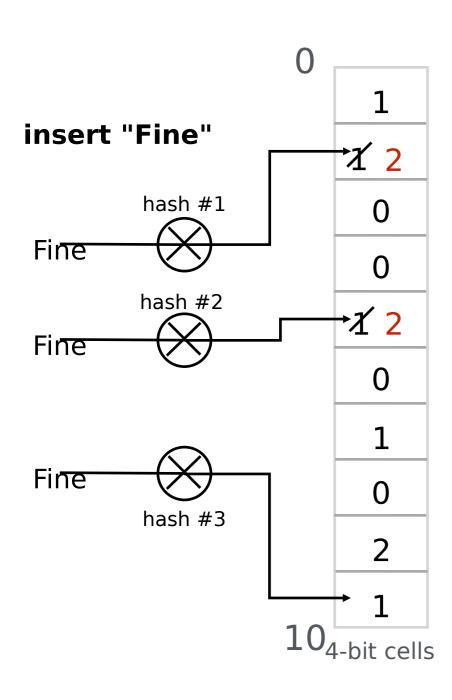
If deleting an element means resetting 1s to 0s, then deleting "Fine" also deletes "TCP"



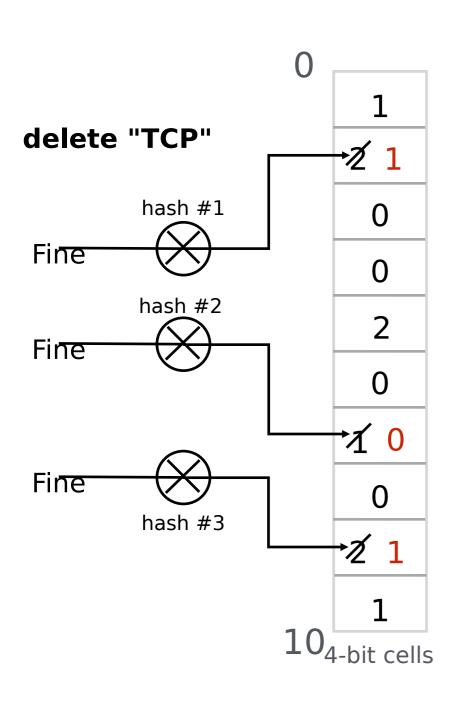
To add an element, increment the corresponding counters



To add an element, increment the corresponding counters

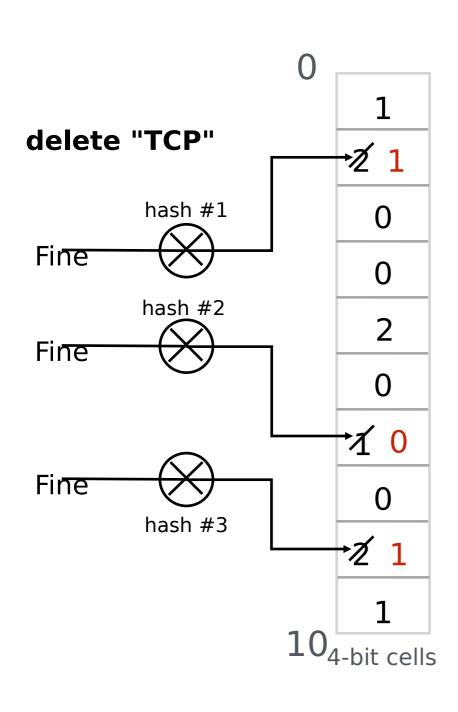


To add an element, increment the corresponding counters



To add an element, increment the corresponding counters

To delete an element, decrement the corresponding counters



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 $M*Pr(Poisson(\ln 2) \geq 2^b) = 1.78\mathrm{e}{-11}$

Add a new element

```
control MyIngress(...) {
    register register<bit<1>>(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 \mid | meta.query2 == 0) {
         bloom_filter.write(meta.index1, meta.query1 + 1);
         bloom_filter.write(meta.index2, meta.query2 + 1);
```

Delete an element

```
control MyIngress(...) {
    register register<bit<1>>(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 Filte	

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

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

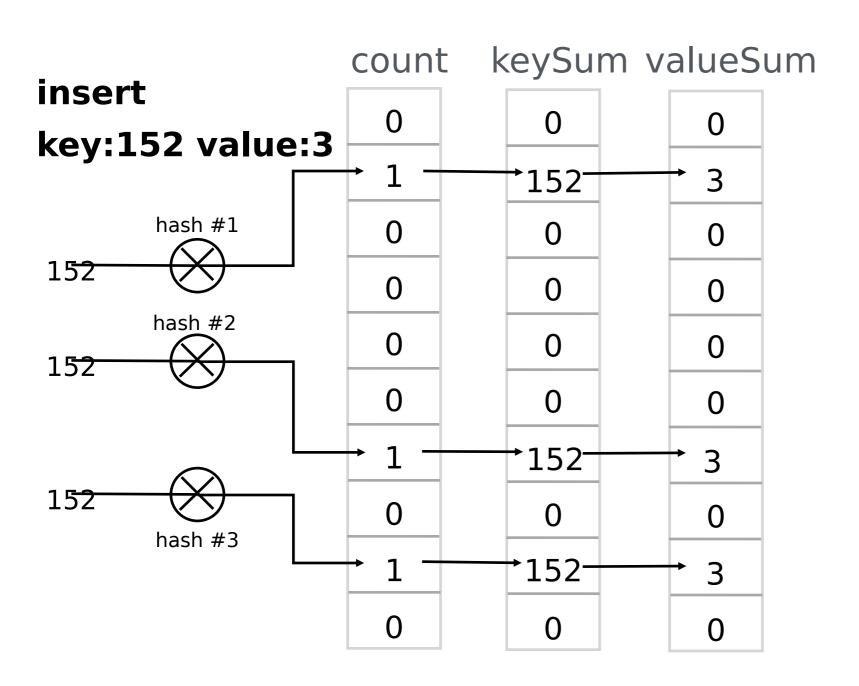
For each hash function hash the key to find the index

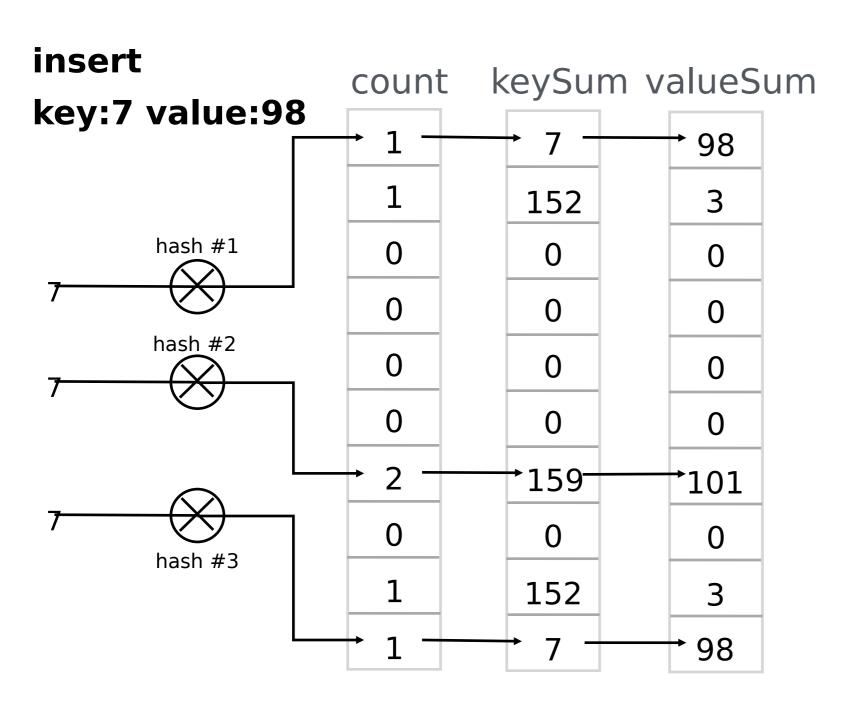
Then at this index increment the count by one add key to keySum add value to valueSum

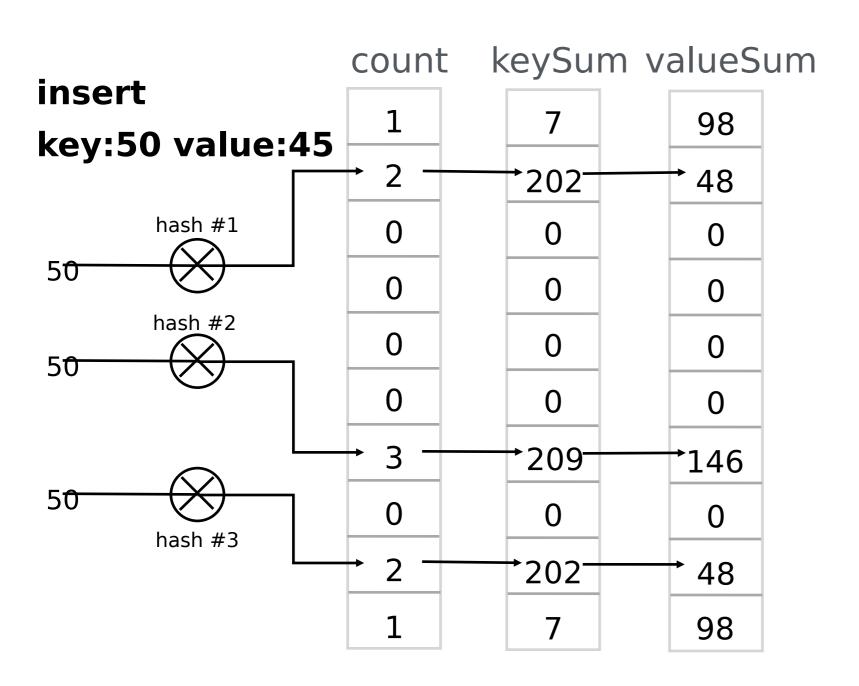
Delete a key-value pair (assuming it is in the set)

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

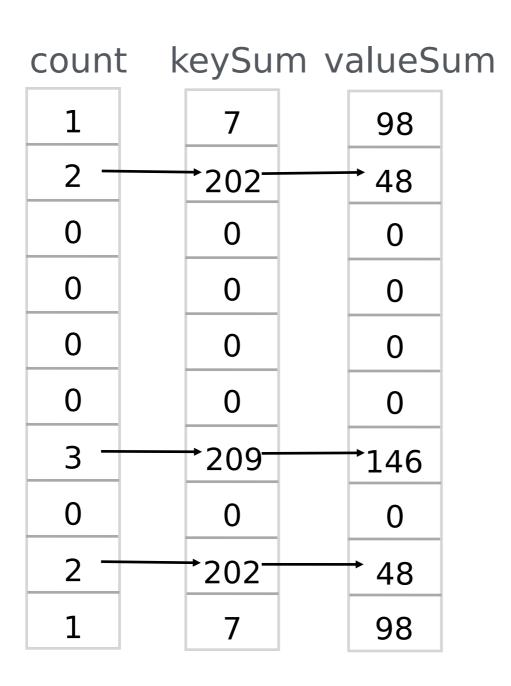


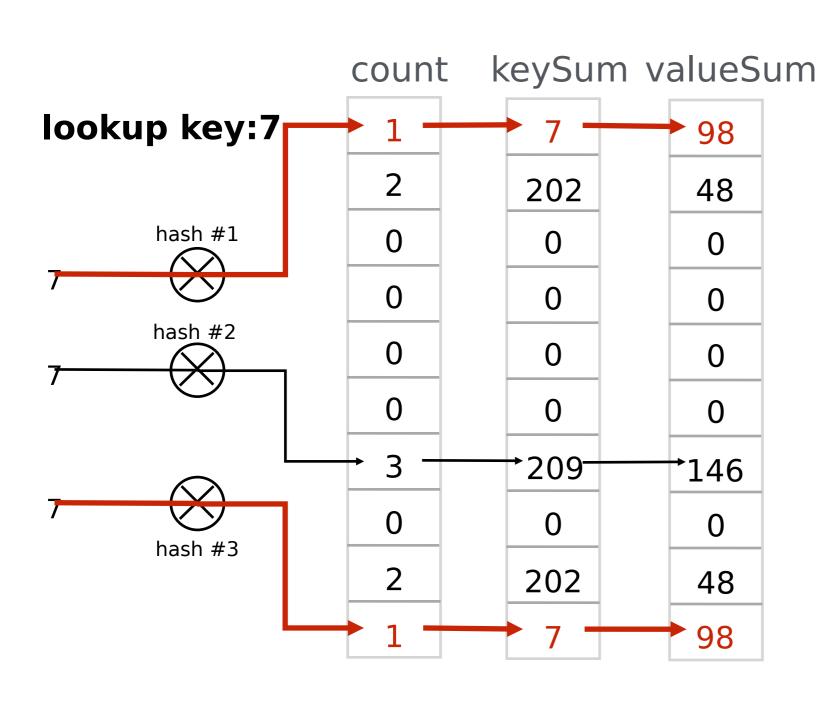




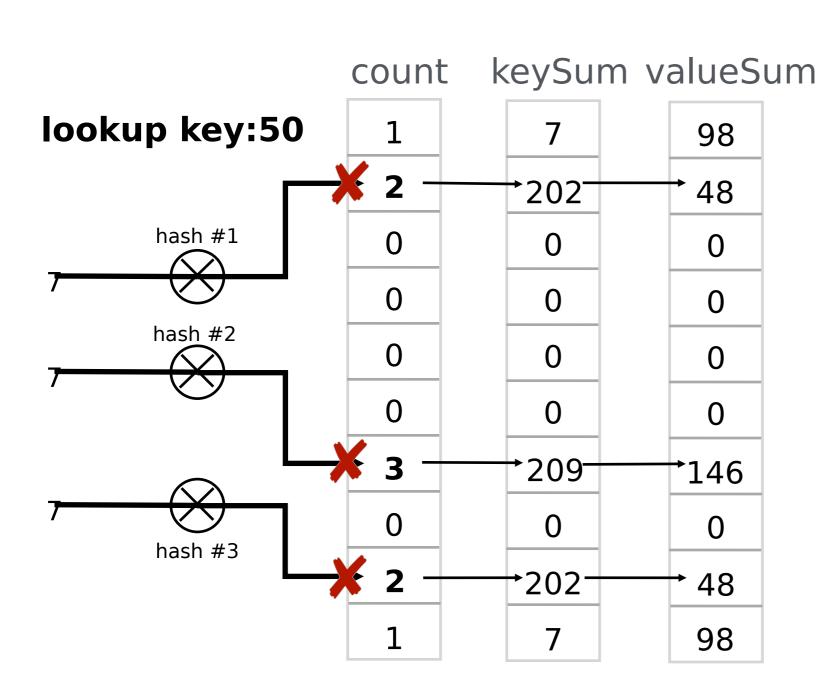
Key-value pair lookup

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





Key 7 has the value 98



The value for the key 50 can't be found

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

Listing the IBLT

While there is an index for which count = 1

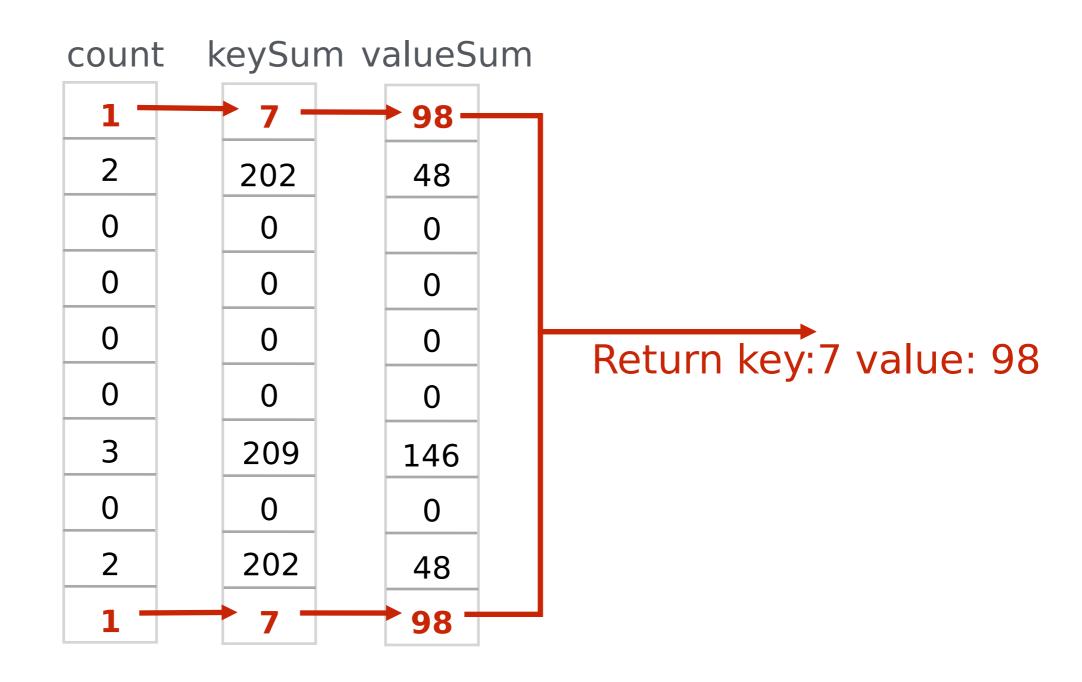
Find the corresponding key-value pair and return it

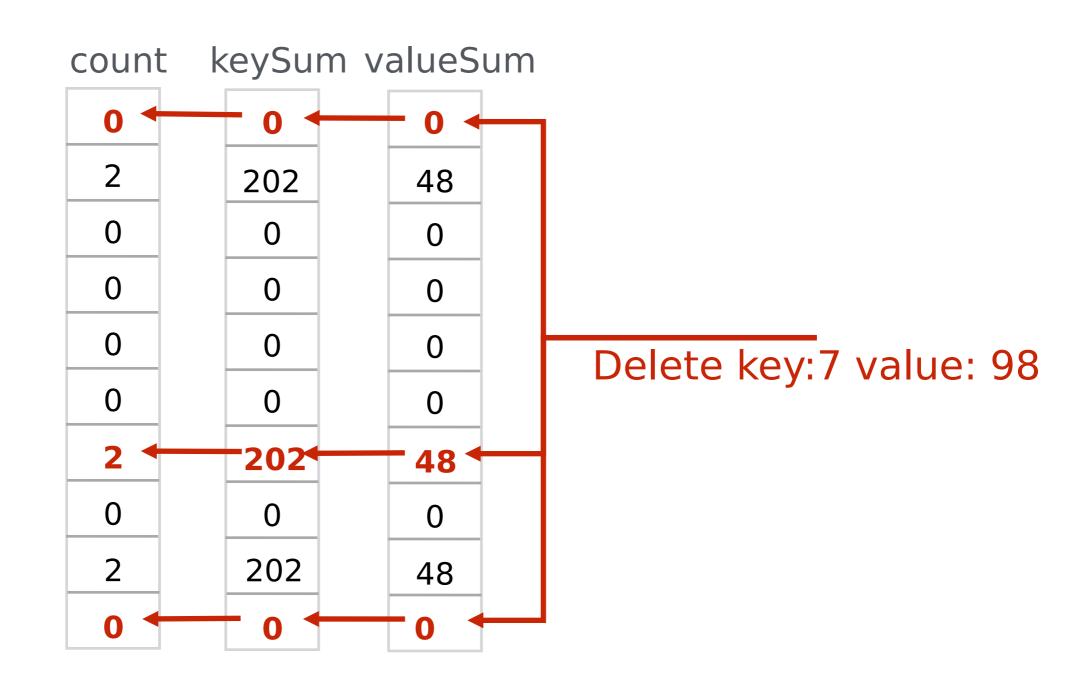
Delete the corresponding key-value pair

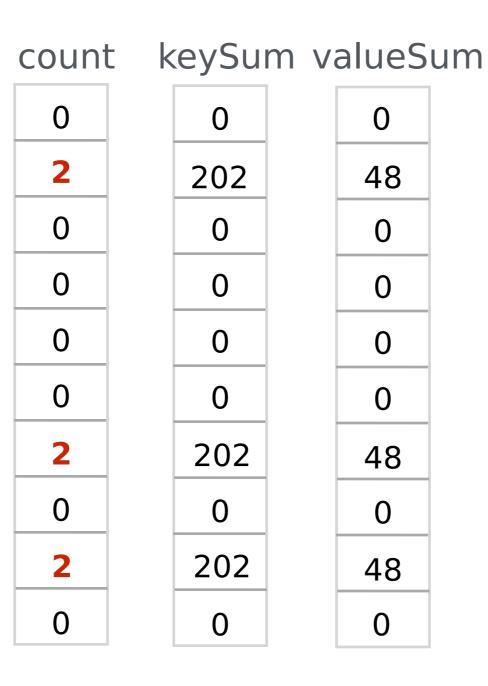
Unless the number of iterations is very low, loops can't be implemented in hardware

The listing is done by the controller

count	keySur	n valueSun
1	7	98
2	202	48
0	0	0
0	0	0
0	0	0
0	0	0
3	209	146
0	0	О
2	202	48
1	7	98

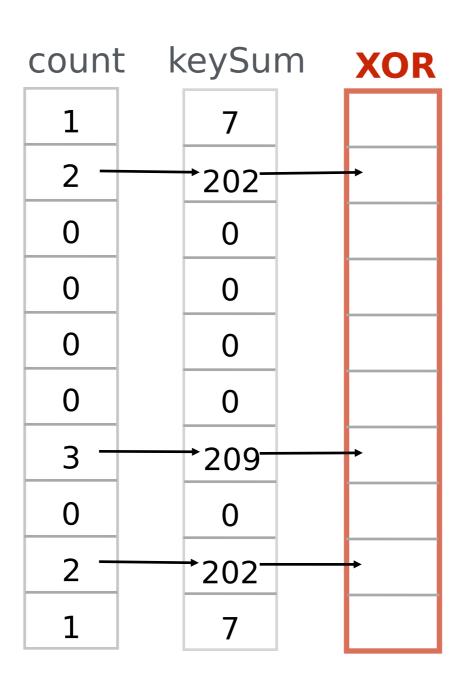






In this example, a complete listing is not possible

In many settings, we can use XORs in place of sums For example to avoid overflow issues



Bloom filters are probabilistic data structures for set membership queries. For more info, see:

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.

You are looking at a stream of data (packets).

Today, I'll show you how set membership and frequency queries can be realized in P4.

Is a certain element (e.g. ip address) in the stream?

→ Bloom filter

PART 2
How frequently does an element appear?

→ CountMin Sketch, Count Sketch, ...

part 2: counting with sketches

How frequently does an element appear?

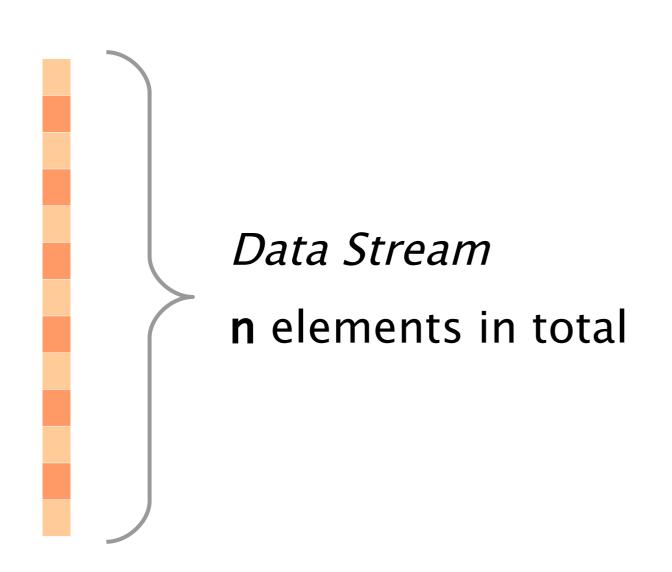
(slides by yours truly)

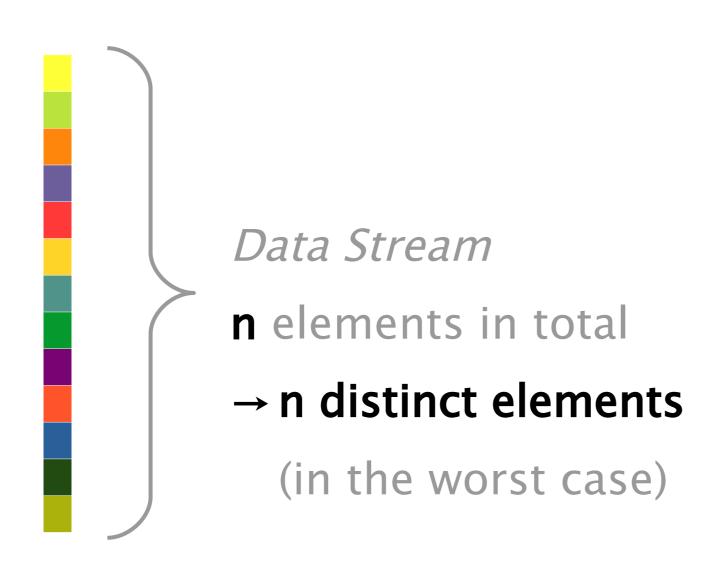
We are going to look at **frequencies**, i.e. **how often** an element occurs in a data stream.

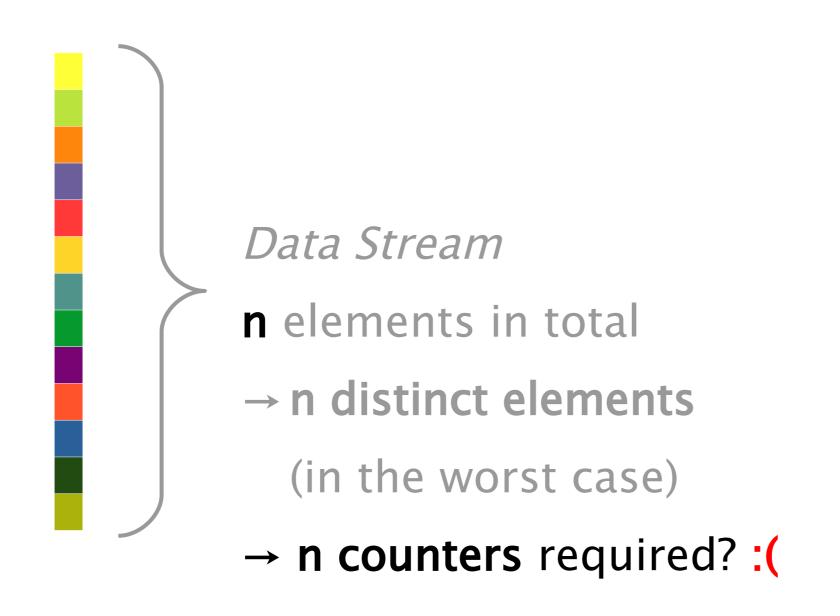
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}$$
 vector of frequencies (counts) of all distinct elements \mathbf{x}_i

We are going to look at **frequencies**, i.e. **how often** an element occurs in a data stream.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}$$
 vector of frequencies (counts) of all distinct elements x_i e.g. flows, ip addresses, ...







Probabilistic datastructures can help again!

Bloom Filters

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

More efficient by allowing false positives.

Probabilistic datastructures can help again!

Bloom Filters

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

More efficient by allowing false positives.

Sketches

provide a approximate frequencies of elements in a data stream.

More efficient by allowing mis-counting.

Notation reminder:

vector of frequencies (counts) of all **distinct elements** \mathbf{x}_i

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

$$Pr \begin{bmatrix} \hat{x}_i - x_i \geq \varepsilon ||x||_1 \end{bmatrix} \leq \delta$$

$$estimated true sum of frequency frequency frequencies$$

The estimation error exceeds $\varepsilon \|x\|_1$ with a probability smaller than δ

 $Pr\left[\begin{array}{c|c} \widehat{x}_i & -x_i & \geq \varepsilon \|\mathbf{x}\|_1 \end{array}\right] \leq \delta$ estimated true sum of frequency frequency frequencies

The estimation error exceeds $\varepsilon \| \mathbf{x} \|_1$ with a probability smaller than δ

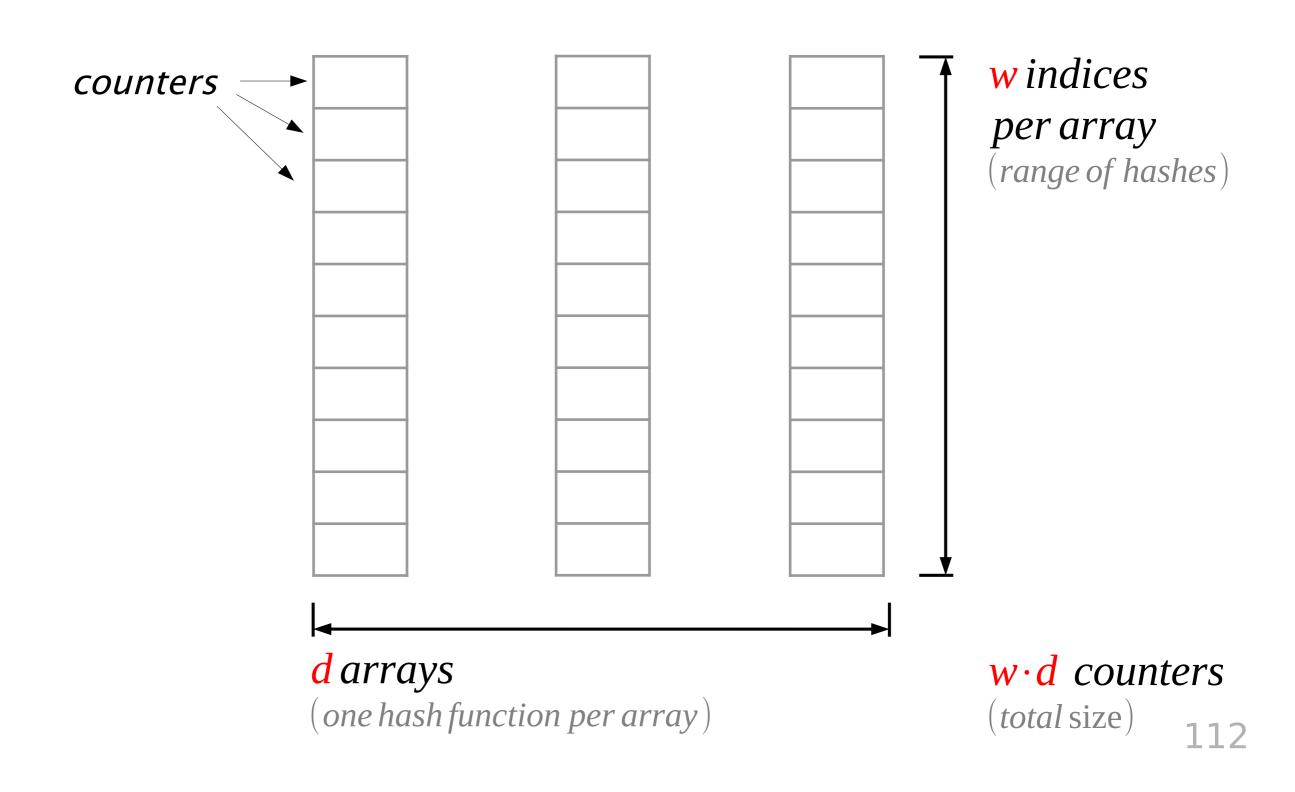
$$Pr \begin{bmatrix} \hat{x}_i - x_i \geq \varepsilon \|x\|_1 \end{bmatrix} \leq \delta$$

$$\text{estimated} \quad \text{true} \quad \text{sum of}$$

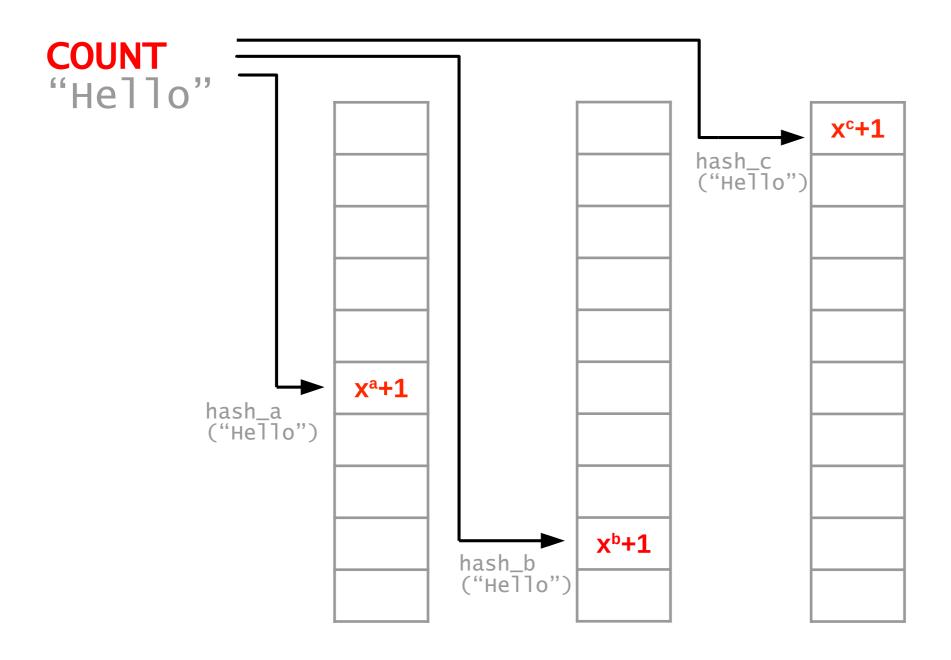
$$\text{frequency} \quad \text{frequency} \quad \text{frequencies}$$

Let $||x||_1 = 10000$, $\varepsilon = 0.01$, $\delta = 0.05$ Then, after counting 10000 elements in total, the probability for any estimate to be off by more than 100 is less than 5%.

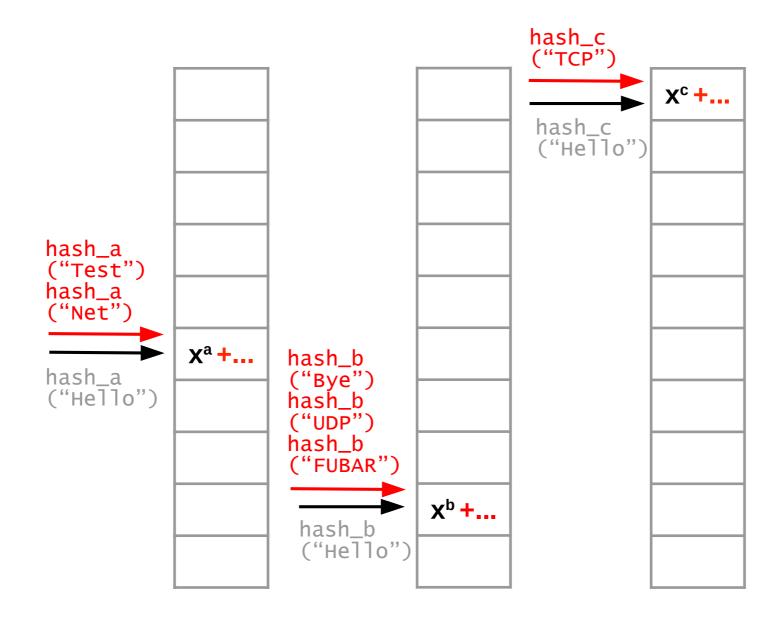
A CountMin Sketch uses multiple arrays and hashes.



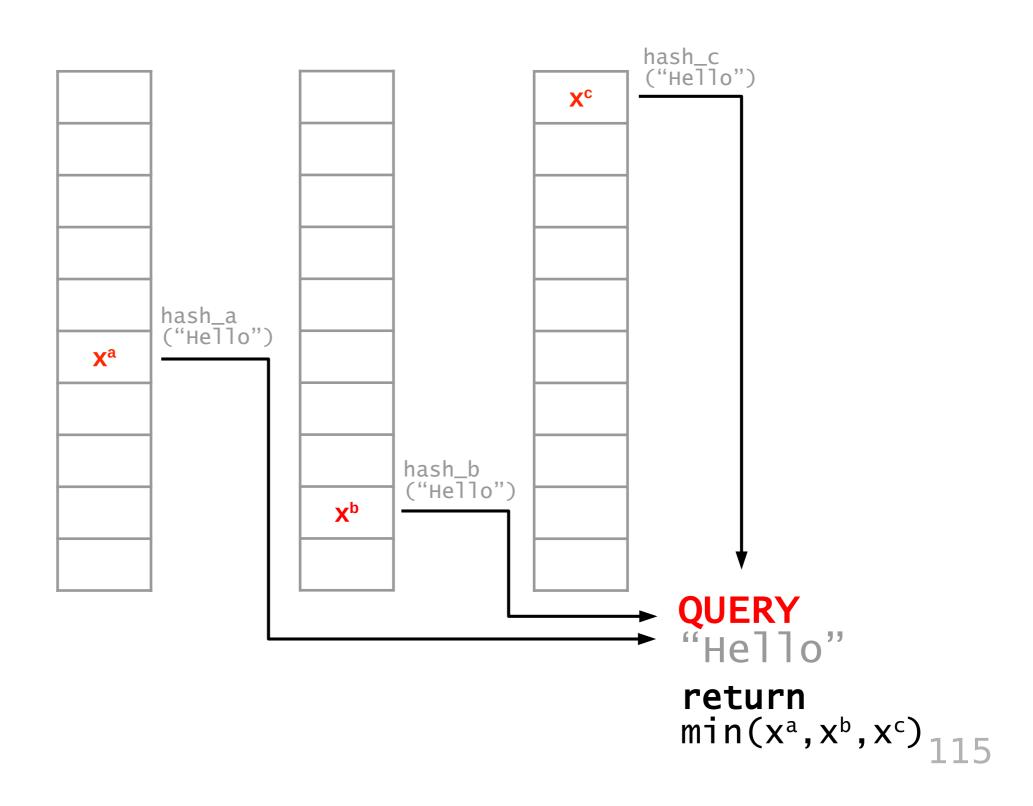
To count, increment all hash-indexed fields by 1.



Hash collisions cause over-counting.

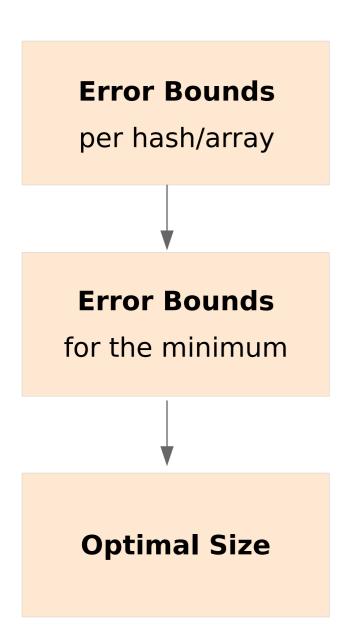


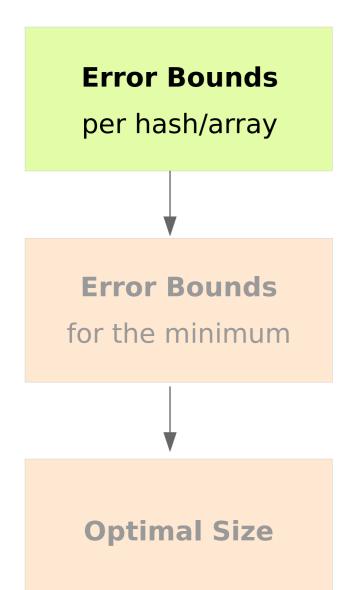
Returning the minimum value minimizes the error.



$$\Pr \left[\begin{array}{ccc} \widehat{x}_i & - & x_i & \geq \varepsilon \|\mathbf{x}\|_1 \end{array} \right] \leq \delta$$
 estimated true sum of frequency frequencies

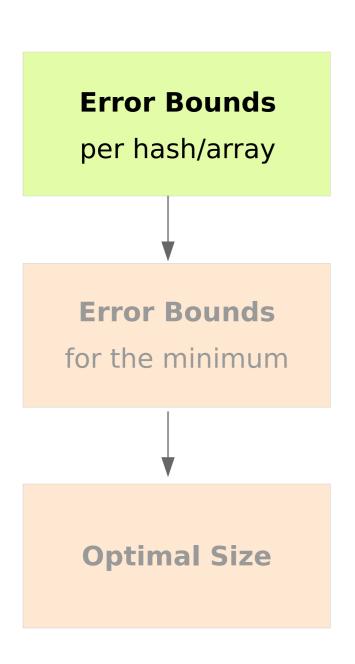
Understanding the error bounds allows **dimensioning** the sketch optimally.





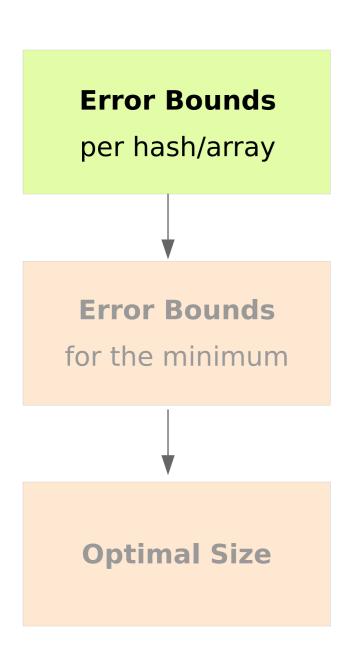
$$\widehat{x}_i = \min_{h \in h_1 ... h_d} \widehat{x}_i^h$$
estimated
frequency
estimate for specific hash

The error bounds can be derived with Markov's Inequality



$$\Pr\left[\mathbf{X} \ge c \cdot E\left[\mathbf{X}\right]\right] \le \frac{1}{c}$$

The error bounds can be derived with Markov's Inequality



$$\Pr\left[\widehat{\mathbf{x}}_{i}^{h} - \mathbf{x}_{i} \ge c \cdot E\left[\widehat{\mathbf{x}}_{i}^{h} - \mathbf{x}_{i}\right]\right] \le \frac{1}{c}$$



Error Bounds

for the minimum

Optimal Size

$$\Pr\left[\widehat{x}_{i}^{h} - x_{i} \ge c \cdot E\left[\widehat{x}_{i}^{h} - x_{i}\right]\right] \le \frac{1}{c}$$

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

true frequency over-counting from hash collisions

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$\Pr\left[\widehat{x}_{i}^{h} - x_{i} \ge c \cdot E\left[\widehat{x}_{i}^{h} - x_{i}\right]\right] \le \frac{1}{c}$$

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

hash collision

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



Error Bounds

for the minimum

Optimal Size

$$\Pr\left[\widehat{x}_{i}^{h} - x_{i} \ge c \cdot E\left[\widehat{x}_{i}^{h} - x_{i}\right]\right] \le \frac{1}{c}$$

$$\widehat{\mathbf{x}}_{i}^{h} - \mathbf{x}_{i} = \sum_{x_{j} \neq x_{i}} x_{j} 1_{h} (x_{i}, x_{j})$$

estimation error

over-counting from hash collisions

Error Bounds

per hash/array

Error Bounds

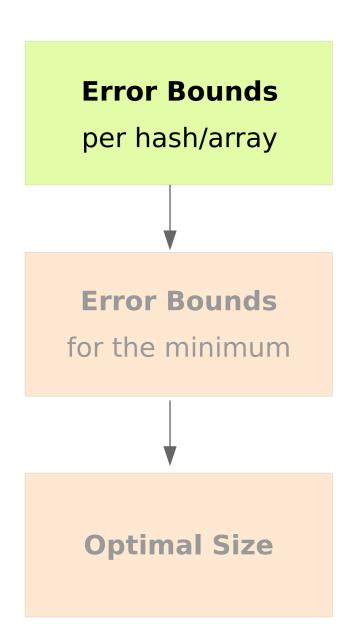
for the minimum

Optimal Size

$$\Pr\left[\widehat{x}_{i}^{h} - x_{i} \ge c \cdot E\left[\widehat{x}_{i}^{h} - x_{i}\right]\right] \le \frac{1}{c}$$

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

$$\underline{E}\left[\widehat{x}_{i}^{h} - x_{i}\right] = E\left[\sum_{x_{j} \neq x_{i}} x_{j} 1_{h}(x_{i}, x_{j})\right]$$

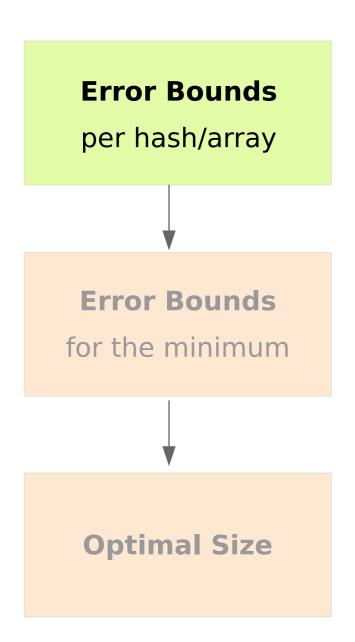


$$\Pr\left[\widehat{x}_{i}^{h} - x_{i} \ge c \cdot E\left[\widehat{x}_{i}^{h} - x_{i}\right]\right] \le \frac{1}{c}$$

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

$$E\left[\widehat{x}_{i}^{h} - x_{i}\right] = E\left[\sum_{x_{j} \neq x_{i}} x_{j} 1_{h}(x_{i}, x_{j})\right]$$

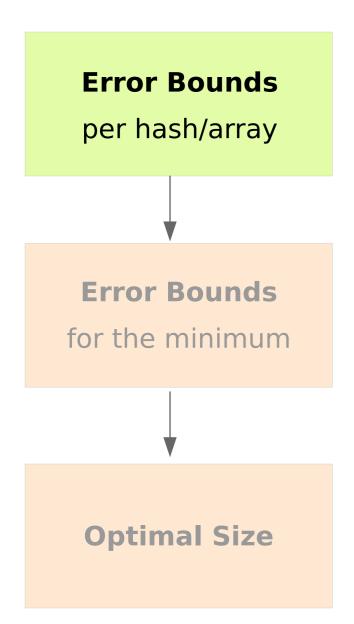
$$\underset{constant}{random}$$



$$\Pr\left[\widehat{x}_{i}^{h} - x_{i} \ge c \cdot E\left[\widehat{x}_{i}^{h} - x_{i}\right]\right] \le \frac{1}{c}$$

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

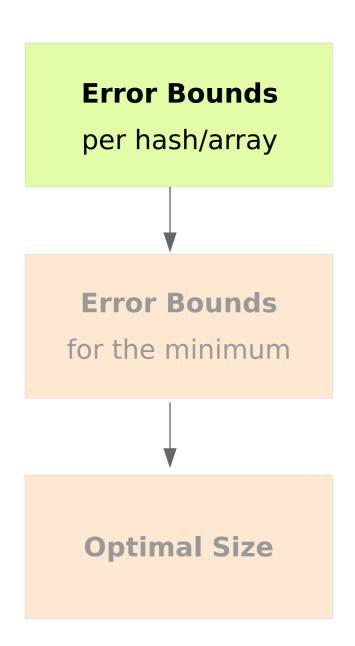
$$E\left[\widehat{x}_{i}^{h}-x_{i}\right] = \sum_{x_{j}\neq x_{i}} x_{j} E\left[1_{h}\left(x_{i}, x_{j}\right)\right]$$



$$\Pr\left[\widehat{x}_{i}^{h} - x_{i} \ge c \cdot E\left[\widehat{x}_{i}^{h} - x_{i}\right]\right] \le \frac{1}{c}$$

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

$$E\left[\widehat{x}_{i}^{h}-x_{i}\right] = \sum_{x_{j}\neq x_{i}} x_{j} \underbrace{E\left[1_{h}\left(x_{i}, x_{j}\right)\right]}_{\leq \frac{1}{w}}$$



$$\Pr\left[\widehat{x}_{i}^{h} - x_{i} \ge c \cdot E\left[\widehat{x}_{i}^{h} - x_{i}\right]\right] \le \frac{1}{c}$$

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

$$\mathrm{E}\left[\widehat{x}_{i}^{h}-x_{i}\right] \leq \sum_{x_{i}\neq x_{i}} x_{j} \frac{1}{w}$$

Error Bounds

per hash/array

Error Bounds

for the minimum

Optimal Size

$$\Pr\left[\widehat{x}_{i}^{h} - x_{i} \ge c \cdot E\left[\widehat{x}_{i}^{h} - x_{i}\right]\right] \le \frac{1}{c}$$

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

$$\mathrm{E}\left[\widehat{x}_{i}^{h} - x_{i}\right] \leq \sum_{\substack{x_{j} \neq x_{i}}} x_{j} \frac{1}{w} \leq \sum_{\substack{x_{j} \neq x_{i}}} x_{j} \frac{1}{w}$$

Error Bounds

per hash/array

Error Bounds

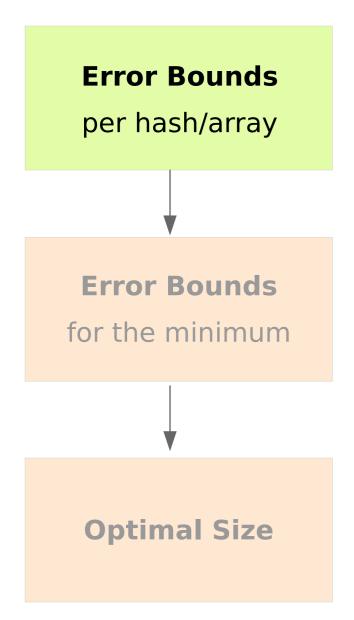
for the minimum

Optimal Size

$$\Pr\left[\widehat{x}_{i}^{h} - x_{i} \ge c \cdot E\left[\widehat{x}_{i}^{h} - x_{i}\right]\right] \le \frac{1}{c}$$

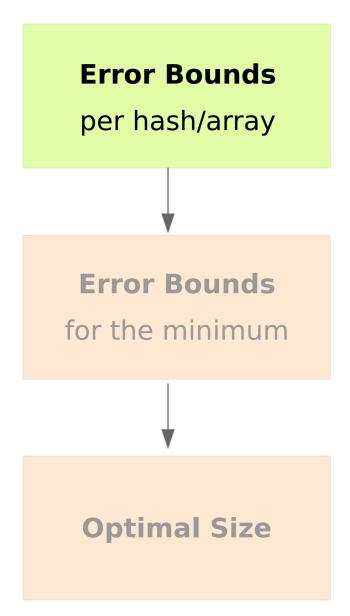
$$\widehat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \, 1_h(x_i, x_j)$$

$$E\left[\widehat{x}_{i}^{h} - x_{i}\right] \leq \sum_{x_{j} \neq x_{i}} x_{j} \frac{1}{w} \leq \left\| \mathbf{x} \right\|_{1} \frac{1}{w}$$

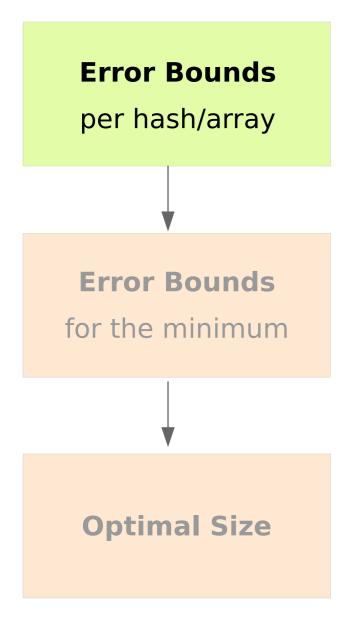


$$\Pr\left[\hat{x}_{i}^{h} - x_{i} \ge c \cdot \underbrace{E\left[\hat{x}_{i}^{h} - x_{i}\right]}\right] \le \frac{1}{c}$$

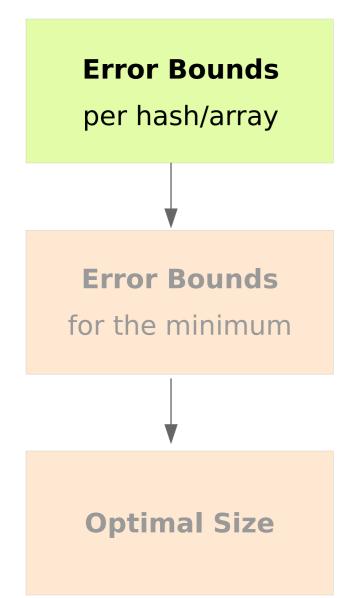
$$\le \frac{1}{w} \|x\|_{1}$$



$$\Pr\left[\widehat{x}_{i}^{h} - x_{i} \ge \frac{c}{w} \|\mathbf{x}\|_{1}\right] \le \frac{1}{c}$$

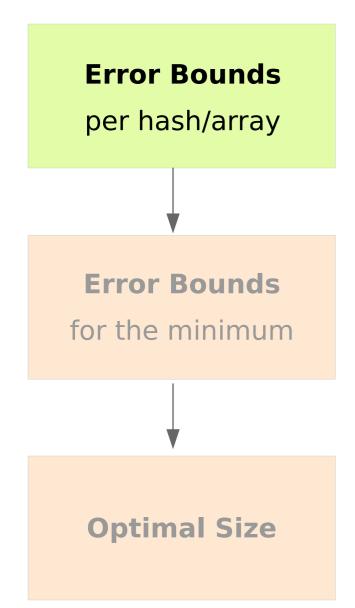


$$\Pr\left[\hat{x}_{i}^{h} - x_{i} \geq \underbrace{\varepsilon}_{w}^{h} \left\| x \right\|_{1}\right] \leq \underbrace{\delta}_{c}^{h}$$



$$\Pr\left[\hat{x}_{i}^{h} - x_{i} \geq \underbrace{\varepsilon}_{w}^{h} \left\| \mathbf{x} \right\|_{1}\right] \leq \underbrace{\delta}_{c}^{h}$$

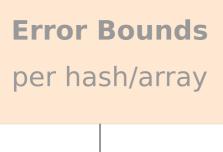
The **estimate for each hash** has a well defined **L1 error bound**.



$$\Pr\left[\widehat{x}_{i}^{h} - x_{i} \geq \underbrace{\varepsilon^{h}}_{w} \left\| \mathbf{x} \right\|_{1}\right] \leq \underbrace{\delta^{h}}_{c}$$

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

What about the minimum?

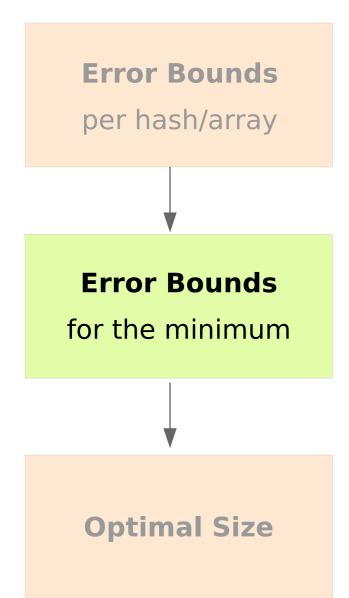


Error Bounds

for the minimum

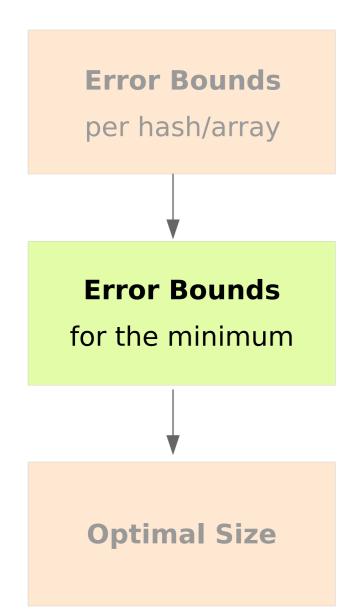
Optimal Size

$$Pr\left[\widehat{\boldsymbol{x}}_{i}-\boldsymbol{x}_{i}\geq\frac{C}{W}\left\|\boldsymbol{x}\right\|_{1}\right] \leq ?$$



$$Pr\left[\min_{\substack{h \in h_1..h_d \\ \hat{X}_i}} \hat{X}_i^h - X_i \ge \frac{C}{W} \|\mathbf{x}\|_1\right] \le ?$$

Multiple hash functions work like independent trials.



$$Pr\left[\min_{\substack{h \in h_1 \dots h_d \\ \hat{x}_i}} \hat{x}_i^h - x_i \ge \frac{c}{w} \|\mathbf{x}\|_1\right] \leq ?$$

$$\Leftrightarrow$$

$$\prod_{h \in h_1 \dots h_d} Pr\left[\hat{x}_i^h - x_i \ge \frac{c}{w} \|\mathbf{x}\|_1\right] \leq ?$$



Error Bounds

for the minimum

Optimal Size

$$Pr\left[\min_{\substack{h \in h_1...h_d \\ \hat{x}_i}} \hat{x}_i^h - x_i \ge \frac{c}{w} \|x\|_1\right] \le ?$$

 \Leftrightarrow

$$\prod_{h \in h_1 \dots h_d} \Pr \left[\hat{x}_i^h - x_i \ge \frac{c}{w} ||x||_1 \right] \le ?$$

error bound per hash



Error Bounds

for the minimum

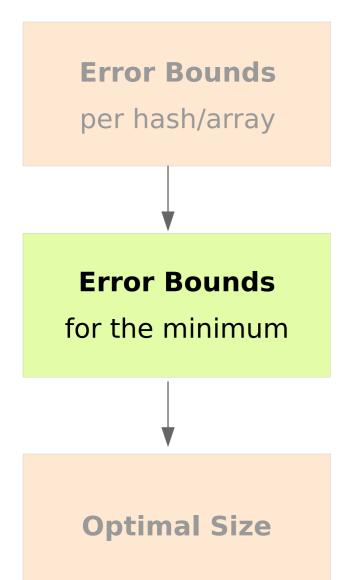
Optimal Size

$$Pr\left[\min_{\substack{h \in h_1...h_d \\ \widehat{X}_i}} \widehat{X}_i^h - X_i \ge \frac{C}{W} \|\mathbf{x}\|_1\right] \le ?$$

$$\Leftrightarrow$$

$$\prod_{h \in h_1 \dots h_d} \Pr\left[\hat{x}_i^h - x_i \ge \frac{c}{w} \|x\|_1\right] \le \frac{1}{c^d}$$

$$\leq \frac{1}{c}$$



$$Pr\left[\min_{\substack{h \in h_1...h_d \\ \widehat{X}_i}} \widehat{X}_i^h - X_i \ge \frac{C}{w} \|\mathbf{x}\|_1\right] \le \frac{1}{c^d}$$

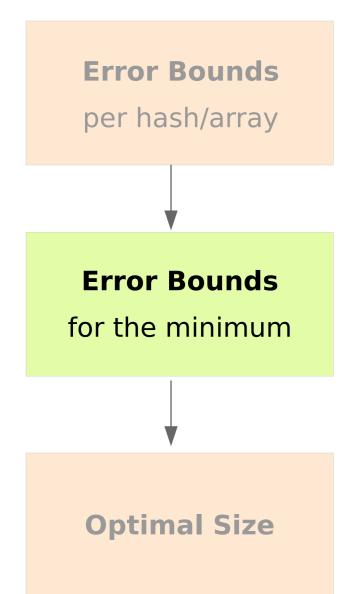


Error Bounds

for the minimum

Optimal Size

$$Pr\left[\widehat{x}_i - x_i \ge \frac{c}{w} \|\mathbf{x}\|_1\right] \le \frac{1}{c^d}$$

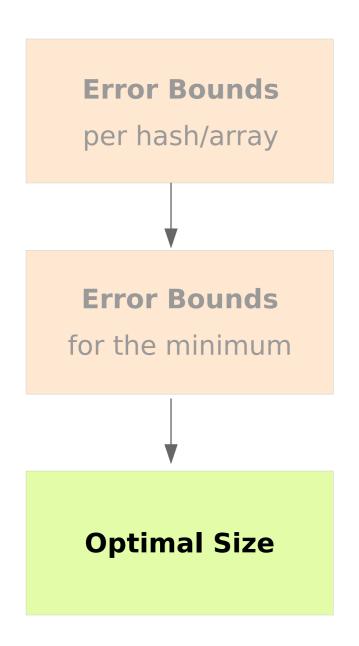


$$Pr\left[\hat{x}_i - x_i \ge \underbrace{\varepsilon}_{w} \|\mathbf{x}\|_1\right] \le \underbrace{\delta}_{\frac{1}{c^d}}$$

We have proven the error bounds!

But what about the constant c?

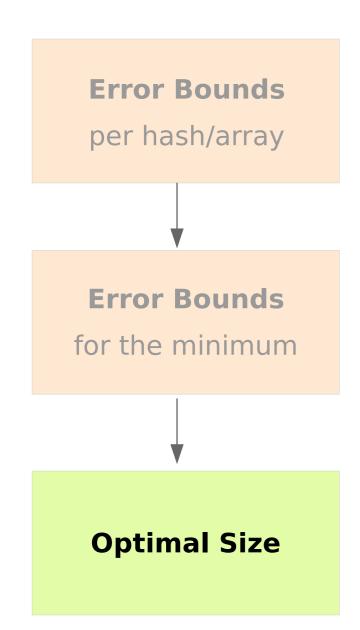
For **every c**, there is a pair (d, w) achieving the error bound and confidence (ε, δ) .



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

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

Choosing c=e minimizes the total number of counters.

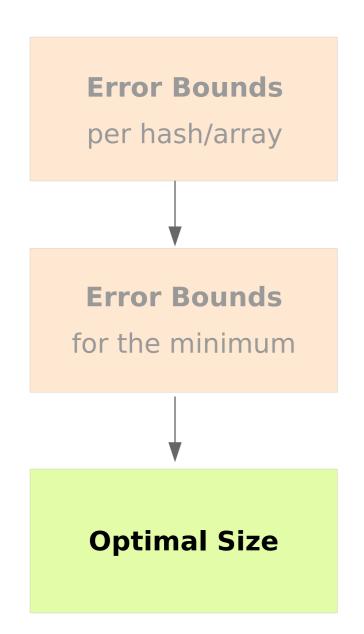


$$\varepsilon = \frac{e}{w} \implies w = \left\lceil \frac{e}{\varepsilon} \right\rceil \qquad (hash range)$$

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

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

A CountMin sketch recipe



Given ε , δ , choosing

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

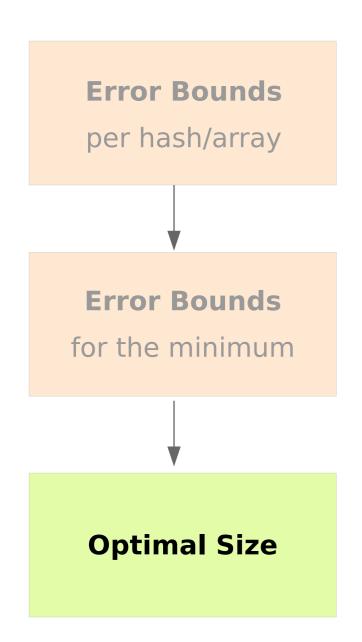
$$d = \left\lceil \ln \frac{1}{\delta} \right\rceil \qquad (\#hashes)$$

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

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

A CountMin sketch recipe

(see extended slides for derivation)



Given ε,δ , choosing

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

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

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

$$\hat{x}_i - x_i \ge \varepsilon \|\mathbf{x}\|_1$$
 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.

CountMin sketch recipe

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

Then $\hat{x}_i - x_i \ge \varepsilon \|\mathbf{x}\|_1$ 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.

```
CountMin sketch
h_1, ..., h_d: U \rightarrow \{1, ..., w\}
COUNT X<sub>i</sub>
for h in h<sub>1</sub>, ..., h<sub>d</sub>:
Reg_h[h(x_i)] + 1
QUERY X;:
return min<sub>h in h1, ..., hd</sub>(
   Reg_h[h(x_i)]
```

CountMin sketch $h_1, ..., h_d: U \rightarrow \{1, ..., w\}$ COUNT X_i for h in h₁, ..., h_d: $\operatorname{Reg}_{h}[h(x_{i})] + 1$ QUERY X;: return min_{h in h1, ..., hd}($Reg_h[h(x_i)]$

```
Count sketch
h_1, ..., h_d: U \rightarrow \{1, ..., w\}
g: U \rightarrow \{+1, -1\}
COUNT X:
for h in h<sub>1</sub>, ..., h<sub>d</sub>:
Reg_h[h(x_i)] + g(x_i)
QUERY X,:
return median<sub>h in h1, ..., hd</sub> (
    Reg_h[h(x_i)] * g(x_i)
```

CountMin sketch recipe

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

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

CountMin sketch recipe

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

Then
$$\widehat{x}_i - x_i \ge \varepsilon \|\mathbf{x}\|_1$$
 with a probability less than δ

Count sketch recipe

Choose
$$d = \left[\ln \frac{1}{\delta} \right], w = \left[\frac{e}{\epsilon^2} \right]$$

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

Sketches are the new black

OpenSketch

NSDI '13

UnivMon

SIGCOMM '16

SketchLearn

SIGCOMM '18

[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 APs for SDN, line attention goes in the measurement and control. While many efforts have been focused on network control APs for SDN, line attention goes in the measurement. The key challenge of designing a new measurement APs 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 achieve (openSketch provides a simple three-stage pipeline (husbing, 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 are mastically configures the pipeline and allocates resources for different measurement tasks. In the control plane, openSketch provides a measurement task in the control plane, openSketch provides a measurement task in the control plane, openSketch provides a measurement task in the control plane, openSketch provides a measurement task in the control plane, openSketch provides a measurement task in the control plane, openSketch provides a measurement task in the control plane, openSketch provides a measurement task in the control plane, openSketch provides a measurement task in the control plane, openSketch provides a measurement task in the control plane, openSketch provides a measurement task in the control plane, openSketch provides a measurement task in the control plane, openSketch provides a measurement task in the control plane, openSketch provides a measurement task in the control plane, openSketch and the control plane, open tasks in the control plane, open

1 Introduction

1 Introduction

Recent advances in software-defined networking (SDN) have significantly improved network management. Network management involves two important stages; to import stages and important stages; to import stages are stages and important stages; to import stages are stages and important stages; to import stages are stages; the Space-Saving stages; the stages are stages; the Space-Saving stages are stages; the Space-Saving stages; the stages are stages; the Space-Saving stages are stages; the Space-Saving stages; the stages are stages; the Space-Saving stages are stages; the Space-Saving stages; the stages are stages; the Space-Saving stages are stages; the Space-Saving stages; the stages are stages; the Space-Saving s

[source]

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

Zaoxing Liu¹, Antonis Manousis⁻, Gregory Vorsanger¹, Vyas Sekar⁻, Vladimir Braverman¹ † Johns Hopkins University * Carnegie Mellon University

Network management requires accurate estimates of metrics for many applications including traffic engineering (e.g.,
heavy hitters), anomaly detection (e.g., entropy of source
addressee), and security (e.g., DDS detection). Obtaining accurate estimates given router CPU and memory constraints is a challenging problem. Estisting approaches fall
in one of two undesirable extreme: (1) low fidelity generalpurpose approaches such as sampling, or (2) high fidelity
but complex algorithms customized to specific applicationlevel metrics. Medily, a solution should be both general
(i.e., supports many applications) and provide accuracy comparable to extend algorithms. This paper presents UnivMon., a framework for flow monitoring which leverages recent theoretical advances and demonstrate that it is possible
to achieve both generally and high accuracy. Univ/hom uses
an application-sponsive data plane monitoring primitive, effferent (and possibly unforescen) estimation algorithms run
in the control plane, and use the statistics from the data plane
for interest that one yield provides resource securacy raised
gentlums and data structures are designed for specific metrics for management as multi-faceted and encompasses a
resource than and many layers and anomaly detection [11,32], attack
and anomaly detection [13], and firencis can plane and timely statistics on different application-level metrics of interest. C.g., the
firent standard provides and provides are supported to a provide and provides are supported to a provide and provides and provides are supported to a provide and provides and provides are supported to a provide and provides and provides are supported to a provide and provides and provides and an alternative class of techniques based on
an application-seponder and provides and provides are supported to a provide and provides and an alternative class of techniques based on
an application-seponder and provides and an alternative class of techniques based on
a provides and provides and provide ferent (and possibly unforescen) estimation algorithms run in the control plane, and use the statistics from the data plane to compute application-level metrics. We present a proof-occopet implementation of UnivMon using P4 and develop simple coordination techniques to provide a "one-by-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 driven evaluations and show that it offers comparable (and

Flow Monitoring, Sketching, Streaming Algorithms

range of tasks including traffic engineering [11, 32], attack and anomaly detection [99], and forerosic analysis [46]. Each such management task requires accurate and timely statistic such management task requires accurate and timely statistic statistics and anomaly detection [97], and forerosic analysis [46]. Each such management task requires accurate and timely statistics are statistically assessed from the statistic statistics and the statistic statistics and the statistic statistics and the statis

in this paper, we present the Univident (short for Universal Monitoring) framework that can simultaneously achieve both generality and high fidelity across a broad spectrum of monitoring tasks [34, 36, 38, 51]. UnivMon builds on and

[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 read deployment. Such user burdens are caused by how existing approximate measurement agreed to the resource accuracy trade of the resource accur 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 a such a such as the confliction of the confliction with accuracy parameters, so as to provision sufficient resources to bound the measurement errors. We despis Selecthicaem, a novel sketch-based measurement framework that resolves resource conflicts by learning their statistical properties to eliminate conflicting traffic components. We prototype Sketchicaem on OpenNyswich and P4, and our testhed experiments and stress-test simulation show that Sketchicaem scarcially monitory accurately and automatically monitory accurately accurate a testore experiments and stress-test simulation show that SketchLearn accurately and automatically monitors various traffic statistics and effectively supports network-wide mea-surement with limited resources.

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

In IRODOCTION

Network measurement is indispensable to modern network management in clouds and data centers. Administrators measure a variety of traffic statistics, such as per-dow 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 statistics to form the basis of management operations such as traffic engineering, performance diagnosis, and intrusion prevention. Unfortunately, measuring traffic statistics is non-trivial or traffic and large-government traffic and large-government traffic and large-government of the proposed for the contraction of the co

tion. Unfortunately, measuring traffic attaitstics 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 [6]. 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, 61], and sketch-based approaches [13, 33, 40, 42, 53], which we collectively refer to as approximate measurement abstractive to record traffic statistics, backed by theoretical guarantees to resource and the statistics, and the statistics of th

Sketches are the new black

LightGuardian

NSDI '21

Nearly-Zero-Error

NSDI '21

CocoSketch

SIGCOMM '21

[source]

LightGuardian: A Full-Visibility, Lightweight, In-band Telemetry System Using Sketchlets

†Department of Computer Science, Peking University, China §Peng Cheng Laboratory, Shenzhen, China ¶Huawei Theory Lab, China

Abstract

Network traffic measurement is central to successful network operations, especially for today's hyper-scale networks. Although existing works have made great contributions, they fail to additive the following these criteria simultaneously:

1) full-stability, which refers to the ability to acquire did to the following three criteria is minutions of the following three criteria is minutions of the contributions of th packet headers. Specifically, we design a novel SuMax sketch to accurately capture flow-level information, SuMax can be divided into sketchlets, which are carried in-band by passing packets to the end-hosts for aggregation, reconstruction, and analysis. We have fully implemented a LightGuardian prototype on a testfed with 10 programmable switches and 8 end-hosts in a FaTfree topology, and conduct extensive experiments and evaluations. Experimental results show that LightGuardian can obtain per-flow per-hop flow-level information within 10 – 1.5 seconds with consistently low overhead, using only 0.07% total bandwidth capacity of the network. using only 0.7% total bandwith capacity of the network. We believe LightGuardian is the first system to collect per-flow per-hop information for all flows in the network with negligible overhead.

Network traffic measurement is central to successful network operations, sepecially for today's hyper-scale networks with more than 10⁴ devices [1-6]. Meanwhile, at end-hosts, knowing the traffic information in the core of the network can also benefit application performance [7-9]. To infer application performance and user experience, the community consensus is to measure at flow-level granularity. Thus, an ideal measurement system is expected to achieve: 1) full-visibility, which we define as the ability to acquire any desired per-hop

¹Co-primary authors: Yikai Zhao, Kaicheng Yang, and Zirui Liu. Cor-

Yikai Zhao†, Kaicheng Yang†, Zirui Liu†, Tong Yang†,§, Li Chen¶, Shiyi Liu[†], Naiqian Zheng[†], Ruixin Wang[†], Hanbo Wu[†], Yi Wang^{‡,§}, Nicholas Zhang[¶]

²Southern University of Science and Technology

flow-level information2 for all flows. Typical desired informa-

- coarsely characterize them into four categories:

 Partial/Sampling solutions [33-17] only sample packets or flows, or collect detailed statistics based on a preconfigured list of conditionals [18, 19]. For instance, Everflow [20] samples each SYN packet, and Cisco switches use the 'match' keyword to specify which network flows need to be counted. Therefore, only a subset of the network traffic is measured with questionable accuracy.

 Probline solutions (6, 2)—24 lineasures the states of devices.
- Probing solutions [6,21–24] measures the states of devices or links by sending probing packets, and only these probes
- on P4-capable switches (§ 2.9).
 In-band solitons carry information in every packer header.
 AM-P94 [\$5] cannot achieve full-visibility with only one bit
 per packet. Although INT [\$6.37] can potentially achieve
 full-visibility, its bandwidth and processing overhead grows
 quickly with the scale of the network. In both the potacard
 [38] (mirroring packets on only the sink switches), the
 number of packers is at least doubled, which is a huge
 barden for the network. *P500 [\$30] uses a cache to group
 packet-level telemetry information according to the flow
 IDs. But its bandwidth coverhead is still morrorious to still morrorious to. IDs. But its bandwidth overhead is still proportional to the

[source]

Toward Nearly-Zero-Error Sketching via Compressive Sensing

Qun Huang^{1,2} Siyuan Sheng³ Xiang Chen^{1,2,4} Yungang Bao³ Rui Zhang⁵ Yanwei Xu⁵ Gong Zhang⁵ ¹Peking University ²Pengcheng Lab ³Institute of Computing Technology, CAS ⁴Fuzhou University ⁵Huawei Theory Department

Abstract

Sketch algorithms have been extensively studied in the area of network measurement, given their limited resource usage and theoretically bounded errors. However, error bounds provided by existing algorithms remain two coarse-grained in practice, only a small number of flows (e.g., heavy hitters) seatment possible error is limited compared to their size, making the practice, only a small number of flows (e.g., heavy hitters) actually benefit from the bunds which the semantice of the practice, only a small number of flows (e.g., heavy hitters) actually benefit from the bunds which the semantice of the provided provided the provided provided the provided provided the provided provided provided the provided provided the provided provi practice, only a shall manifer to flowly (e.g., nearly nations, actually benefit from the bounds, while the remaining flows still suffer from serious errors. In this paper, we aim to design actually benefit of the property of compressive sensing to boost sketch accuracy. Second, we beverage compressive sensing as novel and uniform methodology to analyze various design choices of sketch algorithms that seamlessly embrace compressive sensing to reach nearly zero-cross. We implement our algorithms in OpenViswitch and P4. Experimental results show that the two algorithms incur less than 0.1% per flow error for more than 99.72% flows, while preserving the resource efficiency of sketch algorithms. The efficiency demonstrates the power of our new methodology for sketch analysis and design. tually benefit from the bounds, while the remaining flows

Storch algorithms have been widely adopted in flow-level monitoring. They maintain compact data structures that sear-fice a small portion of accuracy to be readily deployable in commodify network devices. Given their limited overheads and provable high accuracy, unmerous skerch algorithms are designed to monitor various flow statistics, such as per-flow counting [46], heavy hinten [19, 25], denial of-service victims [26, 34] and traffic distributions [46]. These flow statistics from resential building blocks for network management.
Despite the sound theoretical bounds on the errors, existing sketch alterotifisms commiss far from merfect for reconsidirer sketch alterotifisms remain far from merfect for removiding

the maximum possible error is limited compared to their size. Nonetheless, and bound is still unacceptable for most small flows that still suffer from poor accuracy.

In this paper, our goal is to explore nearly-zero-error (NZE) per-flow monitoring. We aim to achieve a negligibly small error (e.g., 29999% flows are reported, and the estimated size of any reported flow has a ch.19 relative error compared to the true size). We base our study on a signal processing technique named compressive sensing, Out key insight is that (1) compressive sensing provides near-perfect signal becovered to the control of the control

addresses the design of NZE sketch, which is never studied. In particular, we exploit compressive sensing in two lines. In the first line, we incorporate the near-perfect recovery tech-nique of compressive sensing by regarding flow statistics at the contribution of the properties of the contribution of the con-tribution to adopt compressive sensing directly. This moti-vates the aecond line of our work that examines the suitability of compressive sensing for schoot algorithms and then de-signs new algorithms accordingly. Specifically, we leverage compressive sensing for spoose a novel and uniform method-ology to study sketch techniques: we formulate various sketch algorithms in forms of matrices and then quantitatively an-alyze their suitability to compressive sensing. Thus, instead of designing from serately, we use the analysis results as a guideline for the algorithm design.

[source]

CocoSketch: High-Performance Sketch-based Measurement over Arbitrary Partial Key Query

Yinda Zhang^{1,3}, Zaoxing Liu², Ruixin Wang¹, Tong Yang¹, Jizhou Li¹, Ruijie Miao¹, Peng Liu¹, Ruwen Zhang¹, Junchen Jiang³

¹Peking University, ²Boston University, ³University of Chicago

Second, they require operators to pre-define the set of flow keys before the measurement start. However, in many use cases, such as network diagnosis and security, it might be difficult to enumerate a few keys that must be measured ahoud of time [21, 42–48]. For instance, DDoS detection may track large flows defined on tens of flow keys including SeCIP-Drift. We brughe, and aristrary preferes before keys required and the start of the superior of the superior of the start of the superior of the start of the

$$Pr \left[\widehat{x}_i - x_i \ge \varepsilon \, || \mathbf{x} ||_1 \right] \le \delta$$
 estimation relative to sum of all elements

Let
$$\varepsilon = 0.01$$
, $||x||_1 = 10000$ $(\Rightarrow \varepsilon \cdot ||x||_1 = 100)$

Assume two flows x_a , x_b ,

high frequency

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

| low frequency

Let
$$\varepsilon = 0.01$$
, $||x||_1 = 10000$ $(\Rightarrow \varepsilon \cdot ||x||_1 = 100)$

Assume two flows x_a , x_b ,

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

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

Let
$$\varepsilon = 0.01$$
, $||x||_1 = 10000$ $(\Rightarrow \varepsilon \cdot ||x||_1 = 100)$

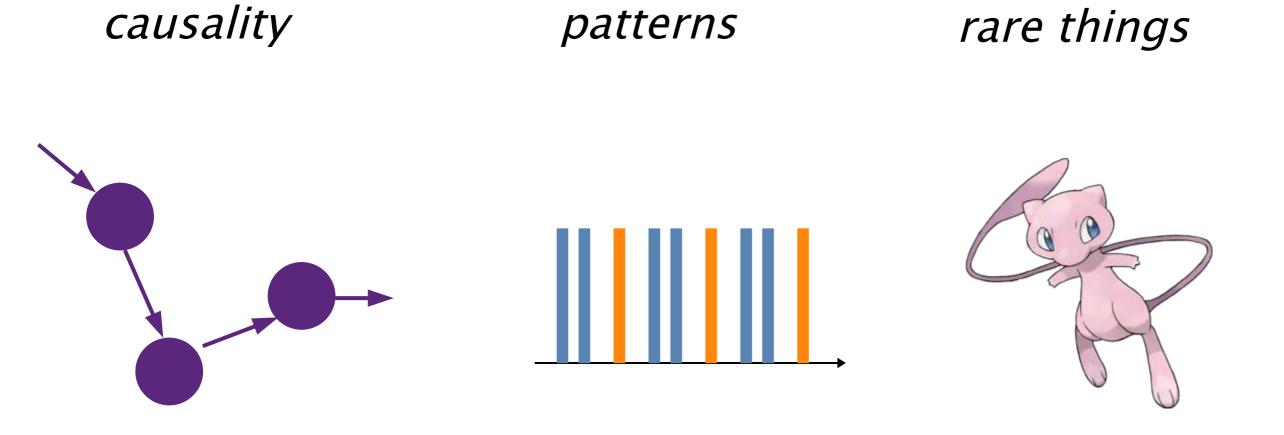
Assume two flows x_a , x_b ,

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

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

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

Other problems a sketch can't handle:



You are looking at a stream of data (packets).
 Today, I'll show you how set membership and frequency queries can be realized in P4.

PART 1
Is a certain element (e.g. ip address) in the stream?

→ Bloom filter

PART 2
How frequently does an element appear?

→ CountMin Sketch, Count Sketch, ...

TAKEAWAY

Probabilistic data structures provide trade-offs between resources and error, and provable guarantees to rely on.