Advanced Topics in Communication Networks

Programming Network Data Planes

Alexander Dietmüller

nsg.ee.ethz.ch

ETH Zürich

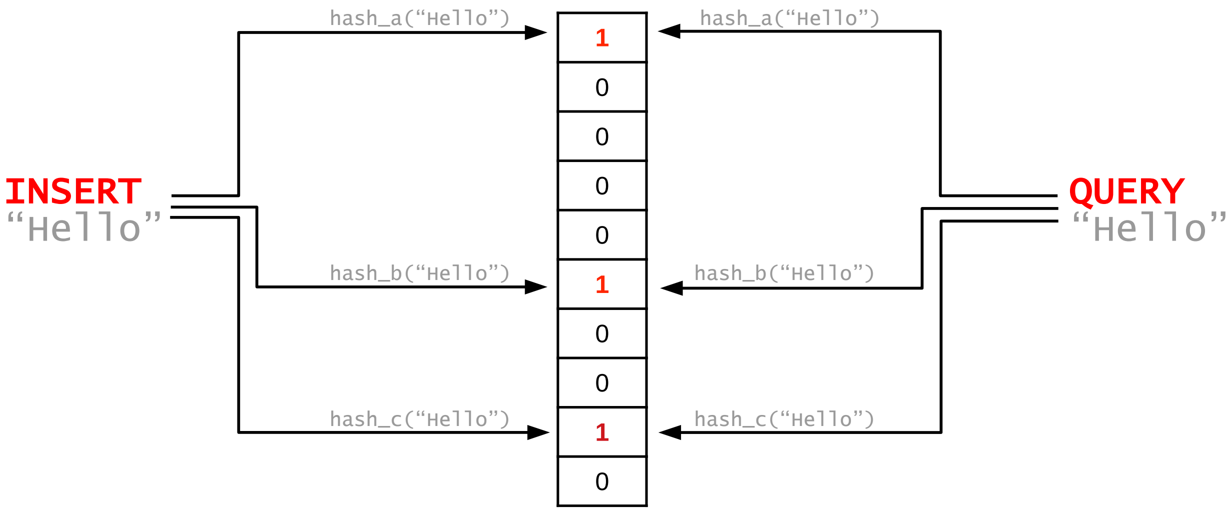
Oct. 11 2018

Last week on

Advanced Topics in Communication Networks

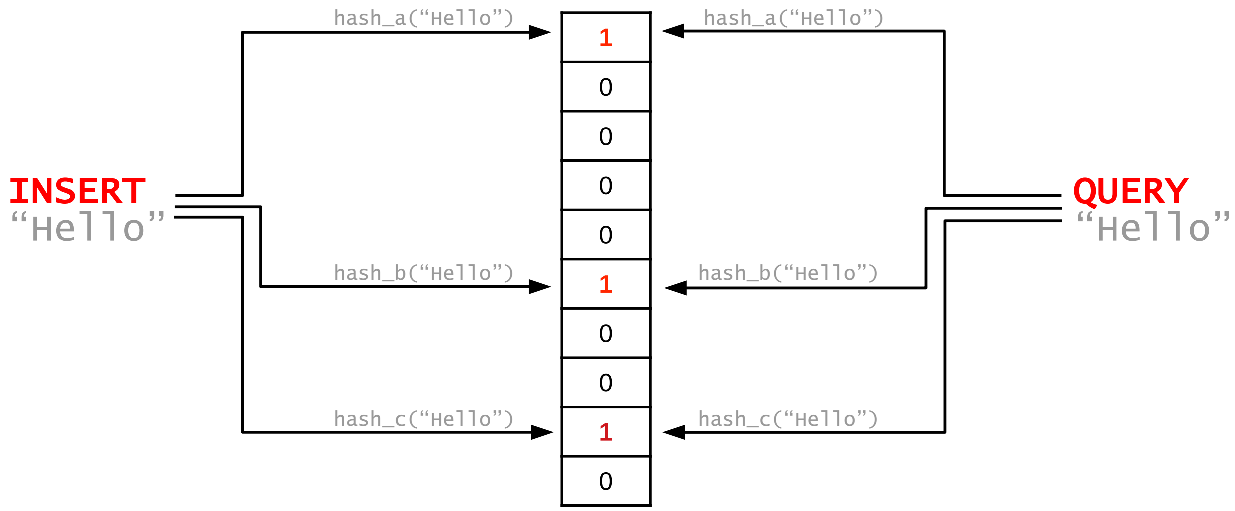
Recap

Probabilistic data structures like **Bloom Filters** help to **trade resources with accuracy**



Recap

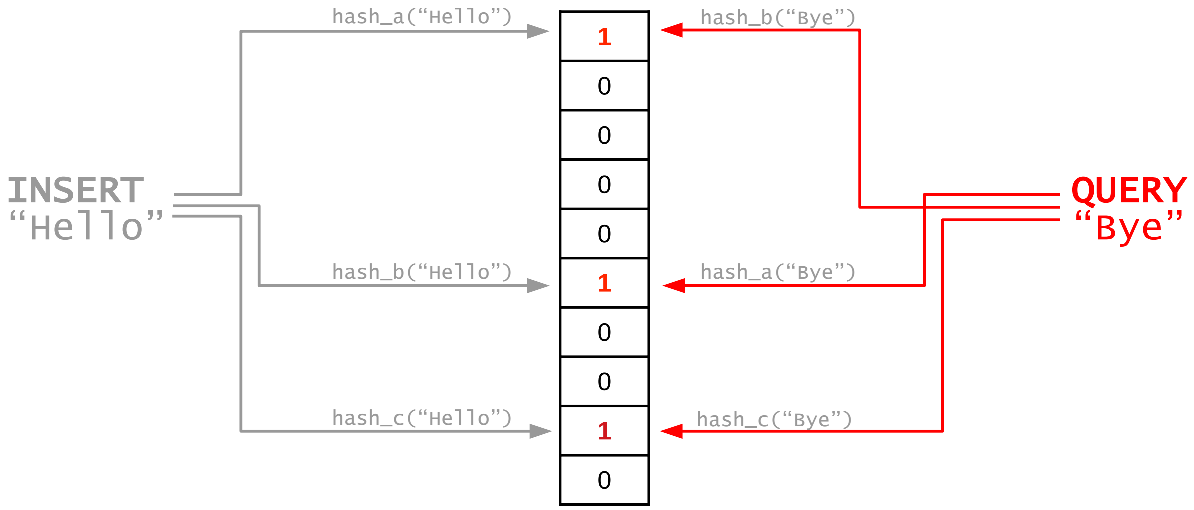
Bloom Filters take a fixed number of operations, but hash collisions can cause false positives.



Recap

Bloom Filters take a fixed number of operations,

but hash collisions can cause **false positives**



Recap

A bloom filter is a streaming algorithm

answering specific questions approximately.

Recap

A bloom filter is a streaming algorithm

answering specific questions approximately.

Is X in the stream?
What is in the stream? Invertible Bloom Filter

A bloom filter is a streaming algorithm

answering specific questions approximately.

Is X in the stream?
What is in the stream? Invertible Bloom Filter

What about other questions?

Today we'll talk about: **important questions,**

how 'sketches' answer them,

and limitations of 'sketches'

my master thesis :)

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.

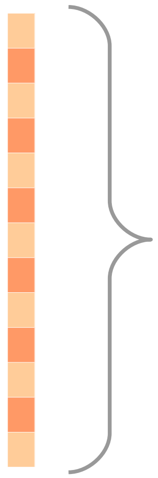
$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}$$

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

distinct flows

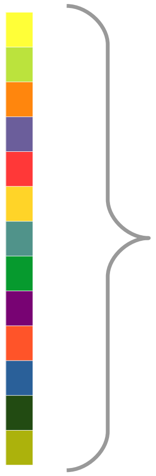
In the worst case, an algorithm providing
exact frequencies requires **linear space**.

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**.



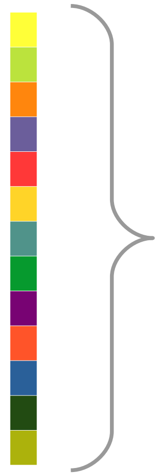
Data Stream

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*

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

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

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

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

relative to L1 norm

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

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

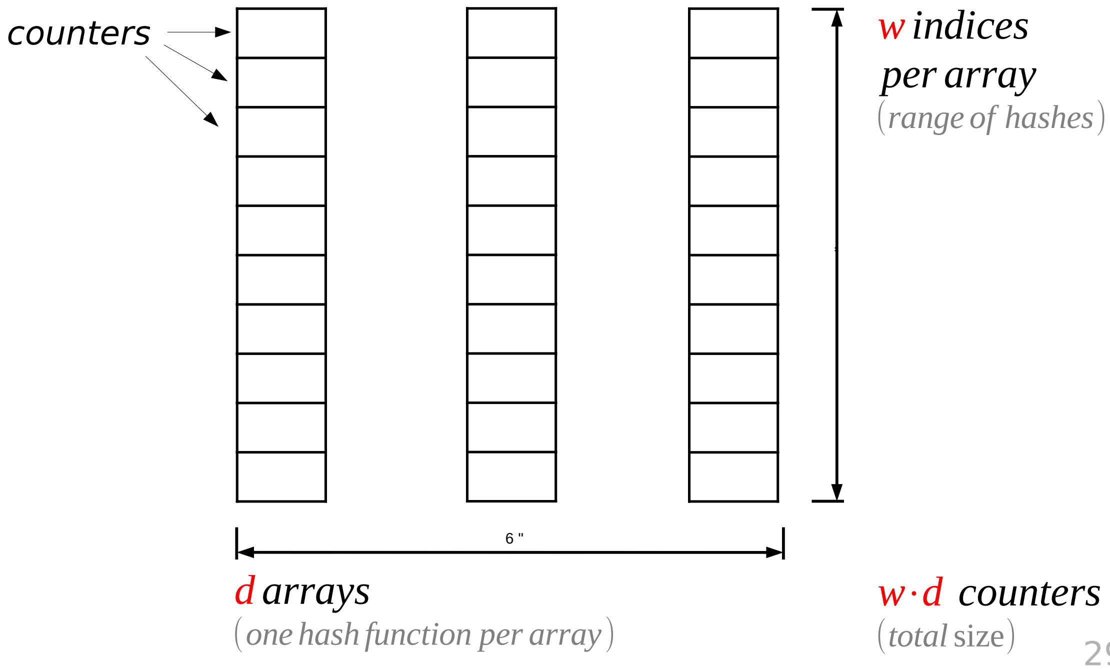
$$Pr[\ \underbrace{\hat{x}_i}_{\text{estimated frequency}} - \underbrace{x_i}_{\text{true frequency}} \geq \varepsilon \underbrace{\|\mathbf{x}\|_1}_{\text{sum of frequencies}}\ ] \leq \delta$$

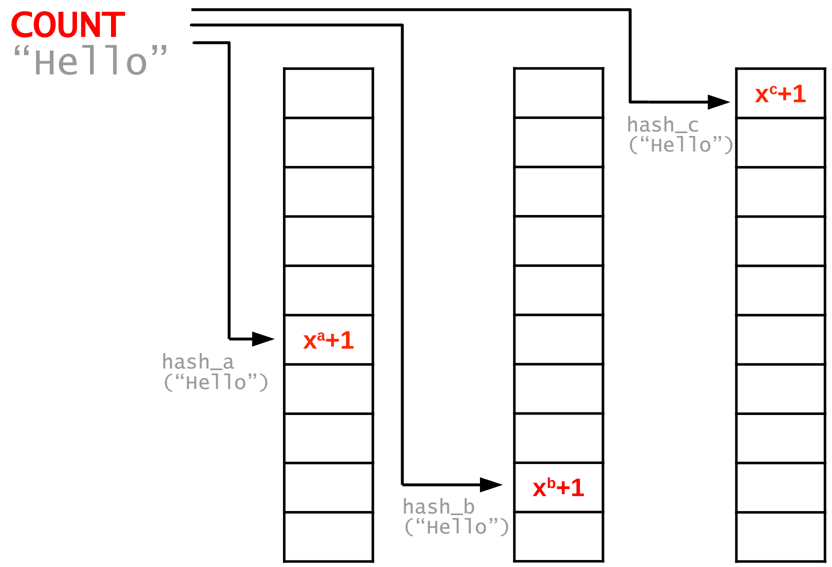
$$Let \quad \varepsilon = 0.01, \quad \delta = 0.05, \quad \|\mathbf{x}\|_1 = 10000$$

The probability for ***any estimate*** *to be off by* ***more than 100*** *is* ***less than 5%***
(after counting 10000 elements)

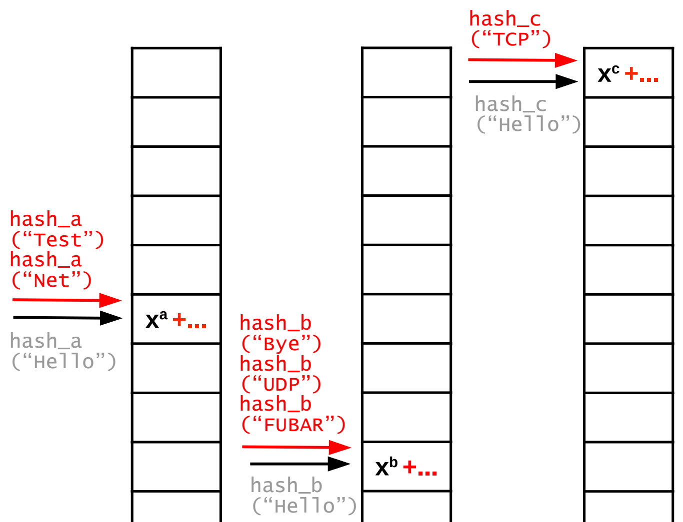
A ***CountMin sketch*** *uses the same principles as a counting bloom filter, but is* ***designed*** *to have* ***provable L1 error bounds*** *for frequency queries.*

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

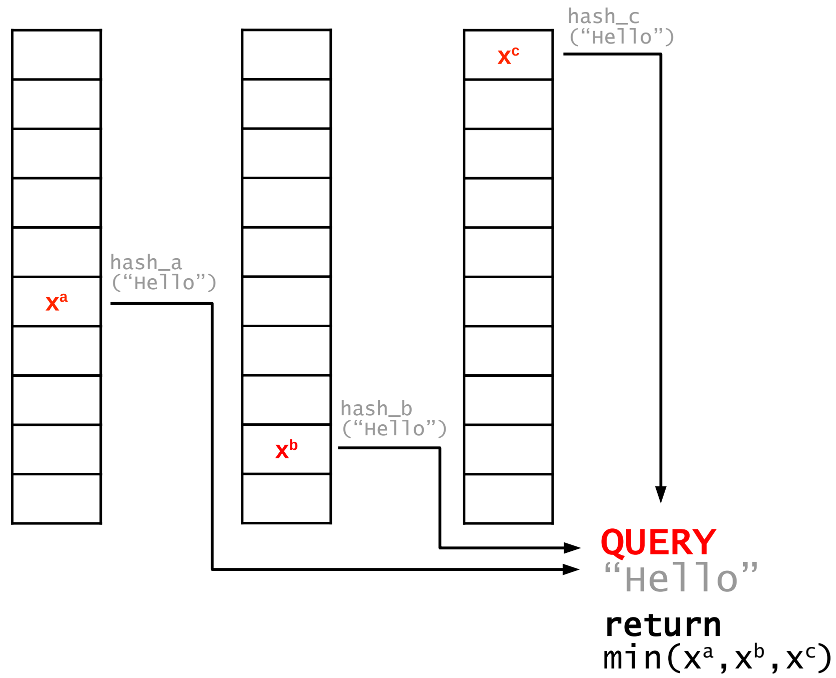




Hash collisions cause **over-counting**.



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.*

$$Pr[\,\hat{x}_i - x_i \geq \varepsilon \|\mathbf{x}\|_1\,] \leq \delta$$

estimated frequency ($\hat{x}_i$) · *true frequency* (x_i) · *sum of frequencies* ($\|\mathbf{x}\|_1$)

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 ($\hat{x}_i$) · *estimate for specific hash* ($\hat{x}_i^h$)

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 ($\hat{x}_i$) · *estimate for specific hash* ($\hat{x}_i^h$)

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

Error Bounds
for the minimum

Optimal Size

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

wikipedia.org/wiki/Markov's_inequality

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

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

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

true frequency

over-counting from hash collisions

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

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

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

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

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

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

estimation error

over-counting from hash collisions

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

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

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

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

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

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

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

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

random

constant

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

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

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

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

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

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

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

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

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

Error Bounds for the minimum

Optimal Size

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

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

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

wikipedia.org/wiki/Universal_hashing

Error Bounds per hash/array

Error Bounds for the minimum

Optimal Size

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

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

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

Error Bounds per hash/array

Error Bounds for the minimum

Optimal Size

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

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

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

Error Bounds per hash/array

Error Bounds for the minimum

Optimal Size

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

Error Bounds per hash/array

Error Bounds for the minimum

Optimal Size

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

Error Bounds per hash/array

Error Bounds for the minimum

Optimal Size

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

Error Bounds per hash/array

Error Bounds for the minimum

Optimal Size

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

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

Error Bounds per hash/array

Error Bounds for the minimum

Optimal Size

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

*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

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

Error Bounds per hash/array

Error Bounds for the minimum

Optimal Size

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

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

Error Bounds per hash/array

Error Bounds for the minimum

Optimal Size

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

$$\Leftrightarrow$$

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

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

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

$$\Leftrightarrow$$

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

error bound per hash

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

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

$$\Leftrightarrow$$

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

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

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

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

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

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

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

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

Error Bounds
for the minimum

Optimal Size

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

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

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

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

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

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

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

A **CountMin** sketch recipe

Error Bounds
per hash/array

Error Bounds
for the minimum

Optimal Size

Given ε, δ***, choosing***

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

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

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

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

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

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

CountMin sketch

```
h_1, …, h_d:   U → {1, …, w}

COUNT x_i:
for h in h_1, …, h_d:
Reg_h[h(x_i)] + 1

QUERY x_i:
return min_{h in h1, …, 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, …, h_d:   U → {1, …, w}

COUNT x_i:
for h in h_1, …, h_d:
Reg_h[h(x_i)] + 1

QUERY x_i:
return min_{h in h1, …, hd}(
   Reg_h[h(x_i)]
)
```

Count sketch

```
h_1, …, h_d:   U → {1, …, w}
g:             U → {+1, -1}

COUNT x_i:
for h in h_1, …, h_d:
Reg_h[h(x_i)] + g(x_i)

QUERY x_i:
return median_{h in h1, …, 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 $\hat{x}_i - x_i \geq \varepsilon \|\boldsymbol{x}\|_1$ *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 $\hat{x}_i - x_i \geq \varepsilon \|\boldsymbol{x}\|_1$ *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 $\hat{x}_i - x_i \geq \varepsilon \|\boldsymbol{x}\|_2$ *with a probability less than* δ

Sketches **are the new black**

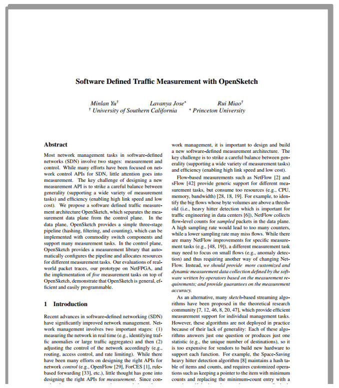
...and many more!

OpenSketch
NSDI ‘13

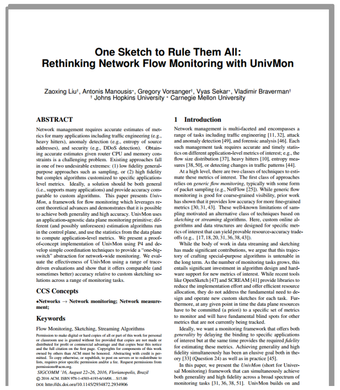
UnivMon
SIGCOMM ‘16

SketchLearn
SIGCOMM ‘18

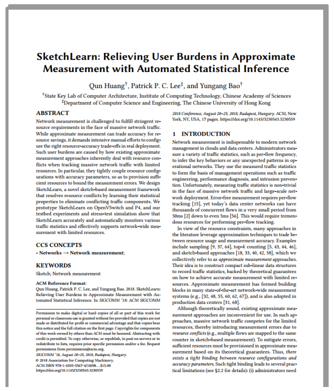
[source]


Software Defined Traffic Measurement with OpenSketch

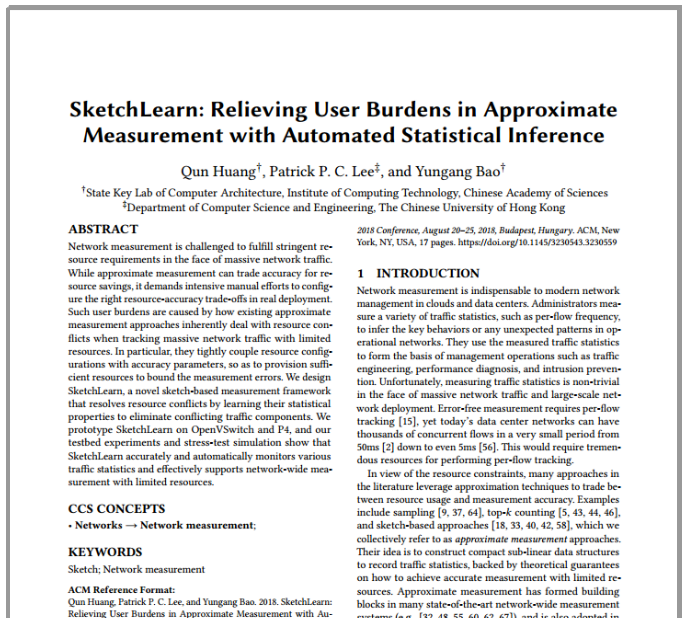
[source]


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

[source]


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

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: Relieving User Burdens in Approximate Measurement with Au-

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

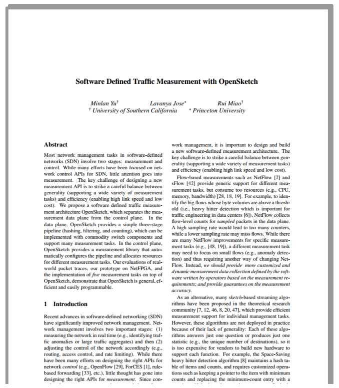
Sketches **are the new black**

OpenSketch
NSDI ‘13

UnivMon
SIGCOMM ‘16

SketchLearn
SIGCOMM ‘18

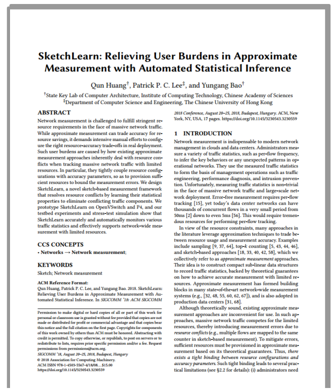
[source]


Software Defined Traffic Measurement with OpenSketch

[source]


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

[source]


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

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**.

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

estimation error

relative to sum of all elements

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

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

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

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

low frequency

high frequency

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

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

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

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

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

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

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

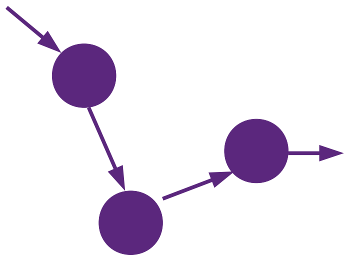
$$\textit{Assume two flows} \quad x_a, \quad x_b,$$

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

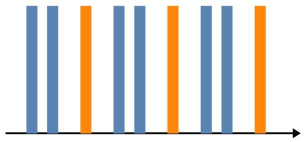
Error relative to **stream size**: 1%
flow size: x_a: 10%, x_b: **200%**

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