Efficient Private Statistics with Succinct Sketches

Luca Melis, *George Danezis, Emiliano De Cristofaro*

University College London

Motivation

- **Gathering statistics in real-world applications**:
  1. Recommender systems for online streaming services
  2. Traffic statistics for the Tor Network

- **Privacy-preserving aggregation can help but…**
  – Protocols do not scale well for large streams

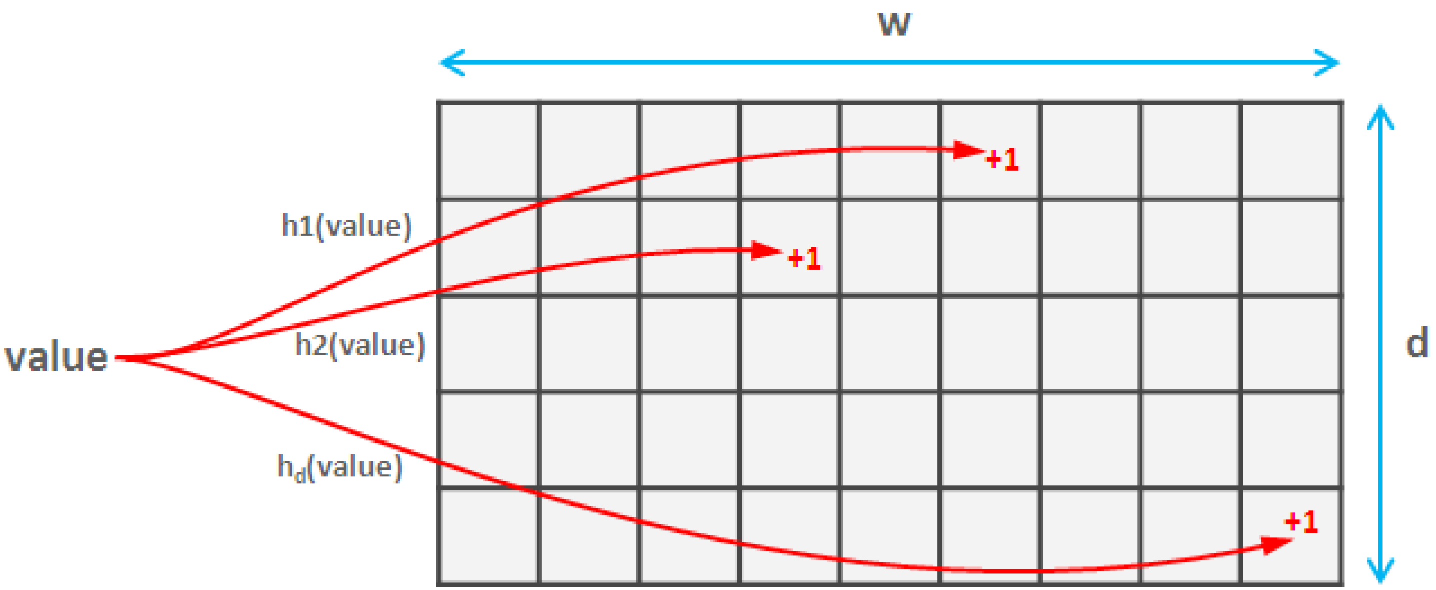
- **Intuition:** Approximate statistics acceptable in some cases for efficiency trade-off

Roadmap

- Privacy-preserving aggregation protocols with "*succinct*" data structures (sketches)
- Reduce complexities from linear to logarithmic in the size of the input streams
- Build practical, easy-to-deploy systems

Preliminaries: Count-Min Sketch

- Estimate item's frequency in a stream by mapping a stream of values (of length T) into a matrix of size O(logT)
- **Key point**: Sum of two sketches yields sketch of the union of the two streams



ItemKNN-based Recommender System

- Predict favorite items for users based on their own ratings and those of “similar” users
- Consider ***N*** users, ***M*** TV programs and binary ratings (viewed/not viewed)
- Build a co-views matrix ***C*****,** where C_{ab} is the number of views for the pair of programs (a,b)
- Compute the **Similarity Matrix**

$$\{Sim\}_{ab} = \frac{C_{ab}}{\sqrt{C_a \cdot C_b}}$$

- Identify K-Neighbours (***KNN***) based on matrix

A Private Recommender System

- Build a global matrix of co-views to train ItemKNN in a privacy-friendly:
  1. Private data aggregation based on secret sharing [Kursawe et al. 2011]
  2. Count-Min Sketch to reduce overhead

- System Model:
  - Users (in groups)
  - Tally Server (e.g, the BBC)

User $\mathcal{U}_i$ $(i \in [1, N])$ — Tally

$$x_i \in_r \mathbb{G}, y_i := g^{x_i} \bmod q \quad \xrightarrow{\quad y_i \quad}$$

$$k_{i_\ell} := \sum_{j \neq i} \mathrm{H}(y_j^{x_i} || \ell || s) \cdot (-1)^{i>j} \bmod 2^{32} \quad \xleftarrow{\quad \{y_j\}_{j \in [1,N]} \quad}$$

$$b_{i_\ell} := X_{i_\ell} + k_{i_\ell} \bmod 2^{32} \quad \xrightarrow{\quad \{b_{i_\ell}\}_{\ell=1}^{L} \quad} \quad \text{Fault recovery (if needed)}$$

$$\xleftarrow{\quad \mathcal{U}^{on} \quad}$$

$$k'_{i_\ell} := \sum_{\substack{j \neq i, \\ j \notin \mathcal{U}^{on}}} \mathrm{H}(y_j^{x_i} || \ell || s) \cdot (-1)^{i>j} \bmod 2^{32} \quad \xrightarrow{\quad \{k'_{i_\ell}\}_{\ell=1}^{L} \quad} \quad C'_\ell := \left(\sum_{i \in \mathcal{U}^{on}} b_{i_\ell} - \sum_{i \in \mathcal{U}^{on}} k'_{i_\ell} \right) \bmod 2^{32}$$

- **Security**
  - Aggregator Obliviousness (AO)
  - Scheme is secure in the honest-but-curious model under the CDH assumption

Implementation

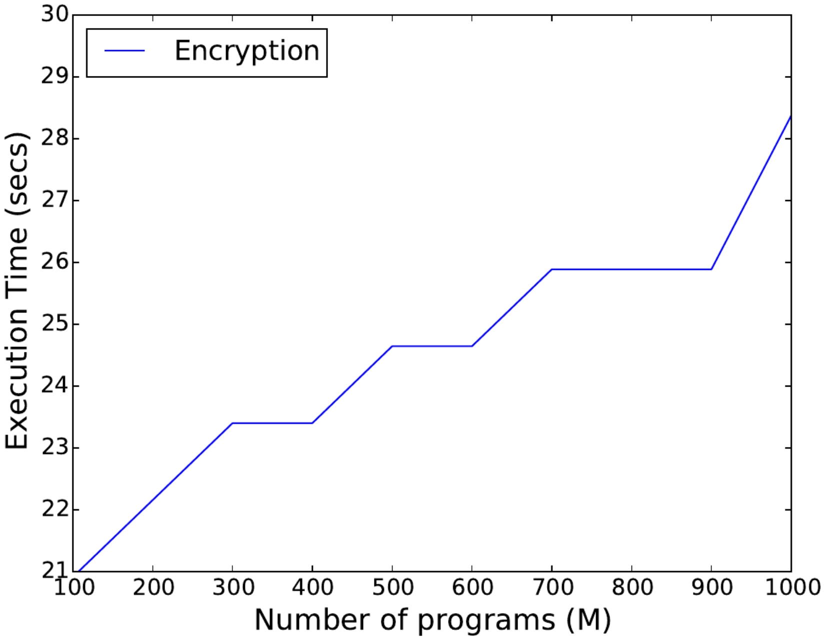
- **Key points**
  - Transparency, ease of use, ease of deployment

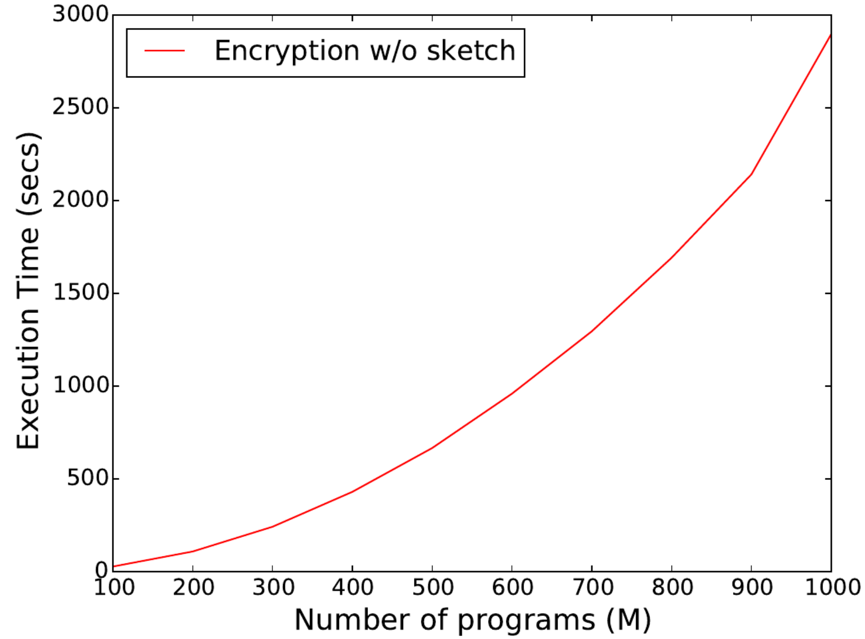
- **Server-side**
  - Tally as a *Node.js* web server

- **Client Side**
  - Runs in the browser
  - Mobile cross-platform application (*Apache Cordova*)

Performance evaluation

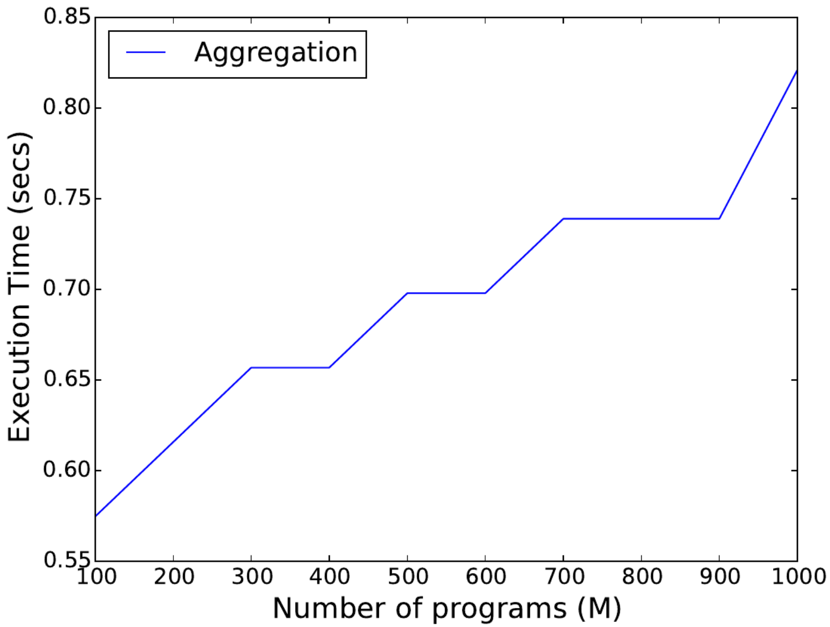
User side (*1,000 users*)

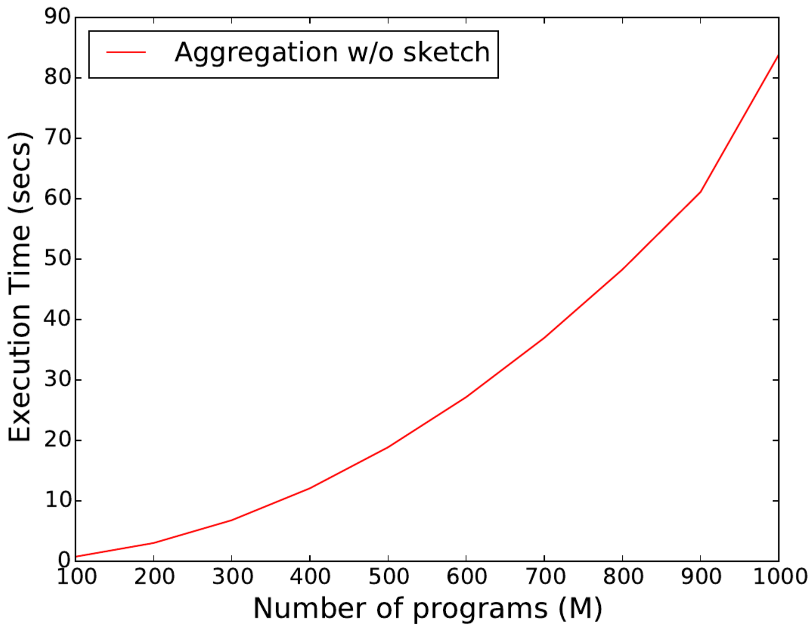


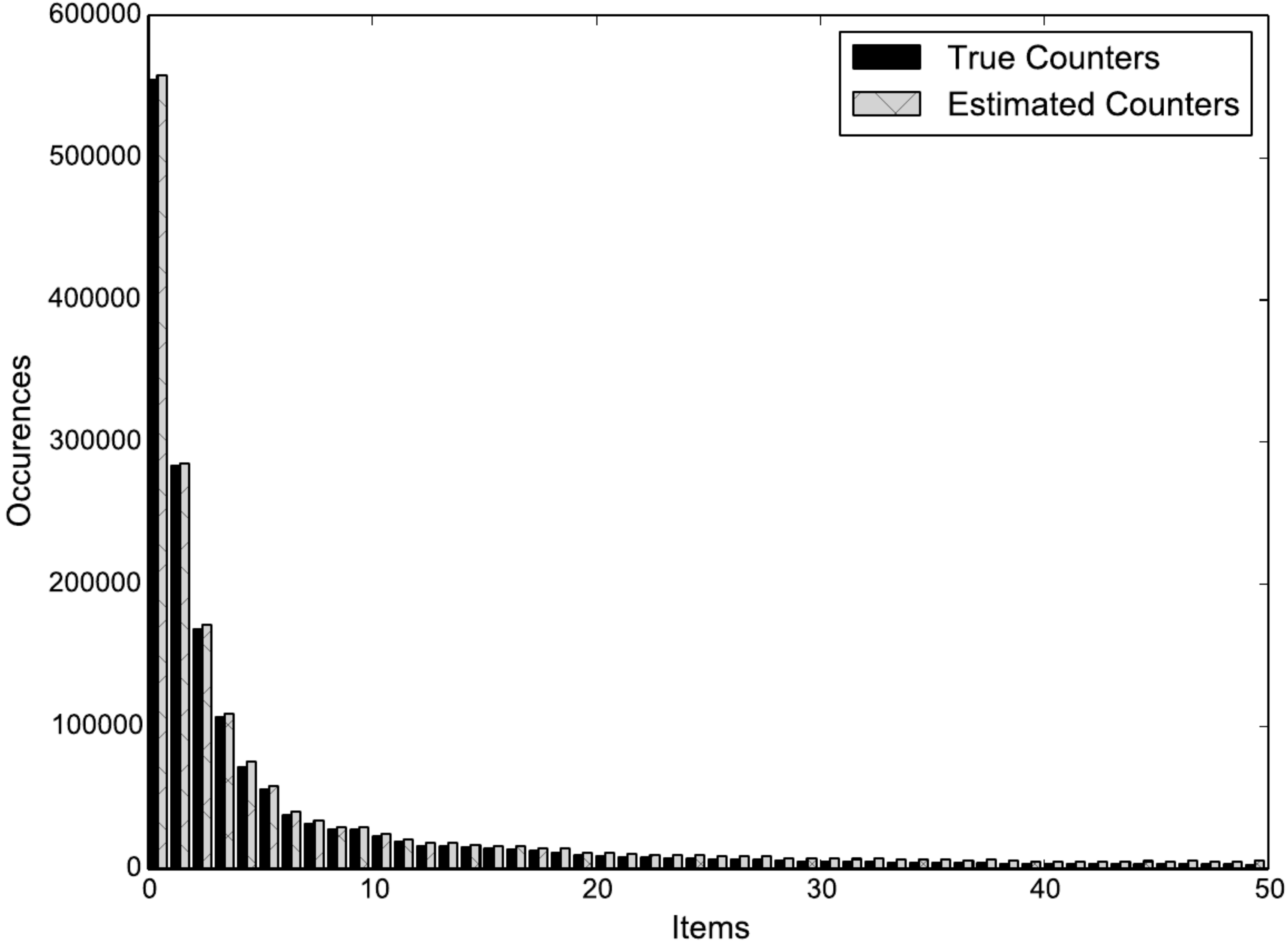


Performance evaluation

Server side (*1,000 users*)






600000
500000
400000
300000
200000
100000
0
Occurences
0
10
20
30
40
50
Items
True Counters
Estimated Counters

Statistics on Tor Hidden Services

- Aggregate statistics about the number of hidden service descriptors from multiple HSDirs
- Median statistics to ensure robustness
- **Problem**: Computation of statistics from collected data can potentially de-anonymize individual Tor users or hidden services

Protocol for estimating median statistics

- We rely on:
  - A set of authorities
  - A homomorphic public-key scheme (AH-ECC)
  - Count-Sketch (a variant of CMS)
- Setup phase
  - Each authority generates their public and private key
  - A group public key is computed

Protocol for estimating median statistics (2)

- Each HSDir (router) builds a Count-Sketch, inserts its values, encrypts it and sends it to a set of authorities
- The authorities:
  - Add the encrypted sketches element-wise to generate one sketch characterizing the overall network traffic
  - Execute a divide and conquer algorithm on this sketch to estimate the median

Estimation of median statistics

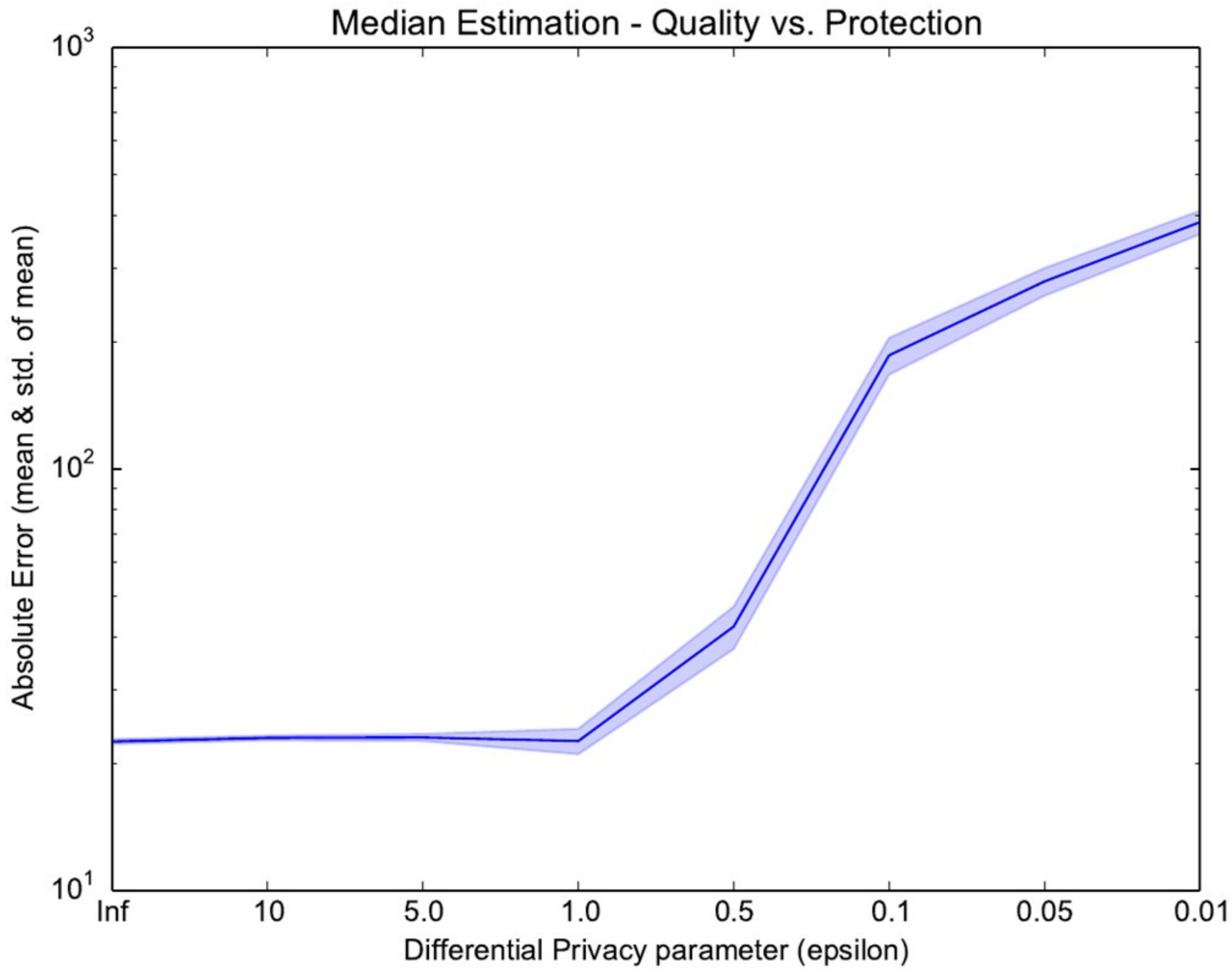
- The range of the possible values is known
- On each iteration, the range is halved and the sum of all the elements on each half is computed
- Depending on which half the median falls in, the range is updated and again halved
- Process stops once the range is a single element
- **Output privacy:**
  - Volume of reported values within each step is leaked
  - Provide *differential privacy* by adding Laplacian noise to each intermediate value

Protocol evaluation

- **Experimental setup:**
  - 1200 samples from a mixture distribution
  - Range of values in [0,1000]

- **Performance evaluation**:
  - Python implementation (*petlib*)
  - 1 ms to encrypt a sketch (of size 165) for each HSDir and 1.5 sec to aggregate 1200 sketches

Quality of estimation vs. privacy protection



Future work

- Apply our private recommender system to news app for Android

- Extend to other machine learning algorithms

- Extend our protocols to malicious security

Thanks for your attention!