Efficient and Robust Data Aggregation Using Untrusted Infrastructure

Marek Klonowski, Michał Koza, Mirosław Kutyłowski

Wrocław University of Technology

SIN 2013,
Aksaray,Turkey

Problem Statement

Model

- Heterogeneous network consist of stations that belong to different owners.

Problem Statement

Model

- Heterogeneous network consist of stations that belong to different owners.
- Typical application - Wireless Sensor Network (WSN)

Problem Statement

Model

- Heterogeneous network consist of stations that belong to different owners.
- Typical application - Wireless Sensor Network (WSN)
- Subset of stations (**subnetwork**) collects some data that should be delivered to the **sink** in an aggregated form.

Problem Statement

Model

- Heterogeneous network consist of stations that belong to different owners.
- Typical application - Wireless Sensor Network (WSN)
- Subset of stations (**subnetwork**) collects some data that should be delivered to the **sink** in an aggregated form.

Problem

- no direct connection between nodes from the subnetwork and the sink ...

Problem Statement

Model

- Heterogeneous network consist of stations that belong to different owners.
- Typical application - Wireless Sensor Network (WSN)
- Subset of stations (**subnetwork**) collects some data that should be delivered to the **sink** in an aggregated form.

Problem

- no direct connection between nodes from the subnetwork and the sink ...
- or sending directly is not efficient (energy necessary for sending for distance r is $\sim r^\nu$ and $\nu > 1$)

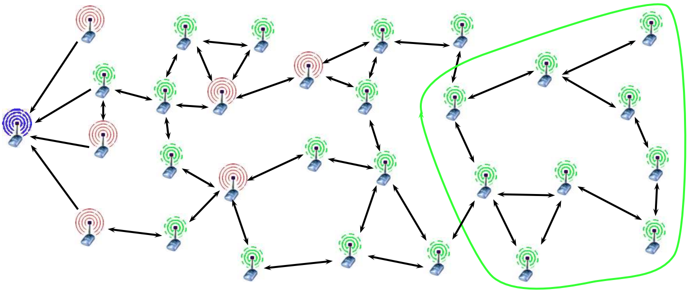
Problem Statement

Model

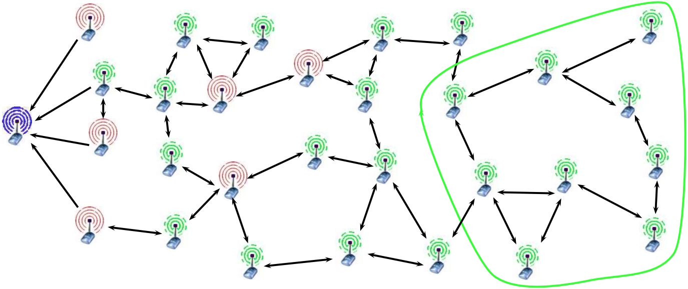
- Heterogeneous network consist of stations that belong to different owners.
- Typical application - Wireless Sensor Network (WSN)
- Subset of stations (**subnetwork**) collects some data that should be delivered to the **sink** in an aggregated form.

Problem

- no direct connection between nodes from the subnetwork and the sink ...
- or sending directly is not efficient (energy necessary for sending for distance r is $\sim r^{\nu}$ and $\nu > 1$)
  $\Rightarrow$ the subnetwork has to use extraneous stations.



Green – trusted stations, try to deliver their values to the sink.



Green – trusted stations, try to deliver their values to the sink. Blue – the sink. Red – extraneous stations, used for data transmission.

Problem Statement - cd

Trust model

Extraneous nodes

- Extraneous nodes are untrusted (adversarial).
- They may aim at:
  - changing the result of data aggregation (even blindly);
  - learning the (partial) result encoded in transmitted data.

Problem Statement - cd
Trust model

Extraneous nodes

- Extraneous nodes are untrusted (adversarial).
- They may aim at:
  - changing the result of data aggregation (even blindly);
  - learning the (partial) result encoded in transmitted data.

Nodes from subnetwork

- Honest-but-curious, should not learn what has been added by other stations.

Problem Statement - cd
Trust model

Extraneous nodes

- Extraneous nodes are untrusted (adversarial).
- They may aim at:
  - changing the result of data aggregation (even blindly);
  - learning the (partial) result encoded in transmitted data.

Nodes from subnetwork

- Honest-but-curious, should not learn what has been added by other stations.
- $\Rightarrow$ only the sink can learn the result of aggregation.

Our protocols

RBF - Robust Bloom Filter

- Result - representation of a subset of elements collected by “green“ stations delivered to the sink.

Our protocols

RBF - Robust Bloom Filter

- Result - representation of a subset of elements collected by “green“ stations delivered to the sink.
- Legitimate stations can add any element.
- Extraneous nodes cannot manipulate the representation. Every manipulation is **detected** with high probability.

Our protocols

RBF - Robust Bloom Filter

- Result - representation of a subset of elements collected by “green“ stations delivered to the sink.
- Legitimate stations can add any element.
- Extraneous nodes cannot manipulate the representation. Every manipulation is **detected** with high probability.
- Protocol supports **idepotence** and **commutativity**. $\Rightarrow$ no synchronization is needed, multi-route processing is possible.
- Only the sink can learn (even the partial) the result.

Our protocols

RCC - Robust Cryptographic Counter

- Better size/security trade-off
- ”Aggregation“ limited to some functions (sum, maximum ...)
- Protocol supports **commutativity** but not **idepotence.**

Bloom Filter - I

- BF is a probabilistic data structure used for representing collection of items.

Bloom Filter - I

- BF is a probabilistic data structure used for representing collection of items.
- We can check if a given element is in the set represented by BF
  - if is, the answer is always TRUE.
  - FALSE positive is possible with some small, **controllable** probability.

Bloom Filter - II

Construction

- depends on two parameters n, l
- a table of n bits; at the beginning all are 0.
- l hash functions $H_1, H_2, \ldots, H_l : \{0,1\}^* \rightarrow \{1, \ldots, n\}$

Bloom Filter - II

Construction

- depends on two parameters n, l
- a table of n bits; at the beginning all are 0.
- l hash functions $H_1, H_2, \ldots, H_l : \{0,1\}^* \to \{1, \ldots, n\}$

Adding element x to BF

1. We compute $\{H_i(x)\}_{i=1}^{l}$
2. $H_i(x)$-th bit of the table is set to 1 for all $1 \leq x \leq l$.

Bloom Filter - Example

Toy Example $n = 9, l = 3$

- Current state of BF is 0 1 1 0 0 1 0 0 1

Bloom Filter - Example

Toy Example $n = 9, l = 3$

- Current state of BF is 0 1 1 0 0 1 0 0 1
- Adding x. We compute $H_1(x) = 8, H_2(x) = 6, H_3(x) = 1$.

Bloom Filter - Example

Toy Example $n = 9, l = 3$

- Current state of BF is 0 1 1 0 0 1 0 0 1
- Adding x. We compute $H_1(x) = 8, H_2(x) = 6, H_3(x) = 1$.
- After that BF is 1 1 1 0 0 1 0 1 1 .

Bloom Filter - Example

Toy Example $n = 9, l = 3$

- Current state of BF is 0 1 1 0 0 1 0 0 1
- Adding x. We compute $H_1(x) = 8, H_2(x) = 6, H_3(x) = 1$.
- After that BF is 1 1 1 0 0 1 0 1 1 .

Is element x' in BF ?

- We compute $H_1(x) = 1, H_2(x) = 7, H_3(x) = 2$ and check if **all** that bits are set to 1.
- 1 1 1 0 0 1 0 1 1

Bloom Filter - Example

Toy Example $n = 9, l = 3$

- Current state of BF is 0 1 1 0 0 1 0 0 1
- Adding x. We compute $H_1(x) = 8, H_2(x) = 6, H_3(x) = 1$.
- After that BF is 1 1 1 0 0 1 0 1 1 .

Is element x' in BF ?

- We compute $H_1(x) = 1, H_2(x) = 7, H_3(x) = 2$ and check if **all** that bits are set to 1.
- 1 1 1 0 0 1 0 1 1

Bound for false positive error

$$< \left(1 - \exp\left(\frac{-l(n + 0.5)}{k - 1}\right)\right)^l .$$

Homomorphic Encryption

Idea

- $E(m_1) \odot E(m_2) = E(m_1 + m_2)$,

Homomorphic Encryption

Idea

- $E(m_1) \odot E(m_2) = E(m_1 + m_2)$,
- Re-encryption without **public** key has to be feasible - every party can re-encrypt a given ciphertext.
- Without the **private** key one cannot distinguish $E(0)$ and $E(1)$.

Homomorphic Encryption

Idea

- $E(m_1) \odot E(m_2) = E(m_1 + m_2)$,
- Re-encryption without **public** key has to be feasible - every party can re-encrypt a given ciphertext.
- Without the **private** key one cannot distinguish $E(0)$ and $E(1)$.
- ElGamal (some other as well).

RBF - Robust Bloom Filter

Idea - I

- aggregated data represented as BF of length n

RBF - Robust Bloom Filter

Idea - I

- aggregated data represented as BF of length n
- trick one - every single bit of BF is represented by a ciphertext of 0 or 1 (homomorphic encryption), called **block.**

RBF - Robust Bloom Filter

Idea - I

- aggregated data represented as BF of length n
- trick one - every single bit of BF is represented by a ciphertext of 0 or 1 (homomorphic encryption), called **block**.
- trick two - we add a m “dummy blocks”- ciphertexts of a fixed value ζ.
  They are randomly permuted with regular blocks.
  Adversary cannot distinguish them. If the dummy block is changed - the sink can detect it.
  $\Rightarrow$ The final result is considered **corrupted**.

RBF - Robust Bloom Filter

Idea - II

- Only stations from the subset knows positions of dummy blocks.
- Such construction (collection of blocks) is traversing the network and **all** blocks are recoded after visiting any "green" station. Moreover the station can add its element to BF (homomorphic property).

RBF - Robust Bloom Filter

Idea - II

- Only stations from the subset knows positions of dummy blocks.
- Such construction (collection of blocks) is traversing the network and **all** blocks are recoded after visiting any “green” station. Moreover the station can add its element to BF (homomorphic property).
- All blocks are delivered to the sink and decoded.
- Very wide class of routing strategies is possible (exact topology of the network may be unknown).

RBF - Robust Bloom Filter

Idea - II

- Only stations from the subset knows positions of dummy blocks.
- Such construction (collection of blocks) is traversing the network and **all** blocks are recoded after visiting any "green" station. Moreover the station can add its element to BF (homomorphic property).
- All blocks are delivered to the sink and decoded.
- Very wide class of routing strategies is possible (exact topology of the network may be unknown).
- Some details are skipped.

Analysis

- trade-off between many values accuracy/security/size/computational complexity
- Hash functions are treated as random oracles.

Analysis

- trade-off between many values accuracy/security/size/computational complexity
- Hash functions are treated as random oracles.

Theorem

If the adversary is capable of computing up to $2\exp(l/2)$ *values of hash functions, then the probability of a successful attack is less then* $(3/4)^l$.

Analysis

- trade-off between many values accuracy/security/size/computational complexity
- Hash functions are treated as random oracles.

Theorem

If the adversary is capable of computing up to $2\exp(l/2)$ *values of hash functions, then the probability of a successful attack is less then* $(3/4)^l$. *False positive do not exceed* $(0,6)^l$. (Some restrictions for m, n and l are omitted.)

Analysis

- trade-off between many values accuracy/security/size/computational complexity
- Hash functions are treated as random oracles.

Theorem

If the adversary is capable of computing up to $2\exp(l/2)$ *values of hash functions, then the probability of a successful attack is less then* $(3/4)^l$*. False positive do not exceed* $(0,6)^l$*.* (Some restrictions for m, n and l are omitted.)

Used techniques

Combinatorial observations, Chernoff bound ...

Analysis
Real values

Weaker

For $k = 60$ elements, size of BF $n + m = 1200 + 600$ and $l = 20$ hash functions

- false positive < 0.00003;
- prob. of successful attack < 0.003 if adversary can compute $< 10^6$ values of hash functions.

Analysis
Real values

Weaker

For $k = 60$ elements, size of BF $n + m = 1200 + 600$ and $l = 20$ hash functions

- false positive < 0.00003;
- prob. of successful attack < 0.003 if adversary can compute $< 10^6$ values of hash functions.

Stronger

For $k = 100$ elements, size $n + m = 10000 + 5000$ and $l = 70$ hash functions

- false positive $< 3 \cdot 10^{-16}$;
- prob. of successful attack $< 1.8 \cdot 10^{-10}$ if adversary can compute $< 10^{17}$ values of hash functions.

RCC - Robust Cryptographic Counter

- Construction is **not** based on BF.

RCC - Robust Cryptographic Counter

- Construction is **not** based on BF.
- The same tricks used in construction.
- We need less blocks (in **some** cases the difference is exponential).

RCC - Robust Cryptographic Counter

- Construction is **not** based on BF.
- The same tricks used in construction.
- We need less blocks (in **some** cases the difference is exponential).
- “Aggregation” is reduced to computing restricted functions e.g. sum or maximum of values.

Thank you for your attention!

marek.klonowski@pwr.wroc.pl