How to Count Travelers Without Tracking Them Between Locations

Nadia Shafaeipour
Faculty of Geo-Information Science and Earth Observation (ITC), University of Twente, Enschede, The Netherlands

Maarten van Steen
Digital Society Institute (DSI), University of Twente, Enschede, The Netherlands

Frank O. Ostermann
Faculty of Geo-Information Science and Earth Observation (ITC), University of Twente, Enschede, The Netherlands

Abstract

Understanding the movements of travelers is essential for sustainable city planning, and unique identifiers from wireless network access points or smart card check-ins provide the necessary information to count and track individuals as they move between locations. Nevertheless, it is challenging to deal with such uniquely identifying data in a way that does not violate the privacy of individuals. Even though several protection techniques have been proposed, the data they produce can often still be used to track down specific individuals when combined with other external information. To address this issue, we use a novel method based on encrypted Bloom filters. These probabilistic data structures are used to represent sets while preserving privacy under strong cryptographic guarantees. In our setup, encrypted Bloom filters offer statistical counts of travelers as the only accessible information. However, the probabilistic nature of Bloom filters may lead to undercounting or overcounting of travelers, affecting accuracy. We explain our privacy-preserving method and examine the accuracy of counting the number of travelers as they move between locations. To accomplish this, we used a simulated subway dataset. The results indicate that it is possible to achieve highly accurate counting while ensuring that data cannot be used to trace and identify an individual.

2012 ACM Subject Classification Security and privacy → Domain-specific security and privacy architectures

Keywords and phrases Privacy preservation, encrypted Bloom filters, traveler counting, subway networks

Digital Object Identifier 10.4230/LIPIcs.GIScience.2023.66

Category Short Paper

Supplementary Material *Software (Source Code)*: `https://github.com/Nadia-Shafaeipour/Counting-travelers-BFs`, archived at `swh:1:dir:58544f9167cea9d5e0f8b973178a59bbca8768aa`

Funding *Nadia Shafaeipour*: This work was supported by Dutch Research Council(NWO).

1 Introduction

As urbanization continues to rise, the usage of public transportation modes is increasing. To implement policies that increase the sustainability of urban transportation systems, a deeper understanding of travel patterns is essential. Traditional surveys and travel diaries require significant effort and provide only snapshots [2]. Various more recent technologies allow automated counting, e.g., Bluetooth and Wi-Fi detection systems and automated fare collection systems. Information gathered by these systems has proved to be helpful in improving security, physical activity, traffic safety, public transportation, communication infrastructure [7, 5], and the overall quality of life for citizens.

However, counting travelers at specific locations by using smart-card IDs allows tracking their movements between locations. It is, therefore, a sensitive issue, especially if it allows monitoring travelers over an extended period of time: the trade-off for valuable insights into movement patterns is an infringement upon their privacy. It has been shown that a few points are enough to identify individual travelers with simple anonymization and persistent identifiers [4].

To prevent such situations, various regulations have been adopted, including Europe's General Data Protection Regulation (GDPR) [6], which requires parties to obtain explicit consent before collecting and using personal information. Obtaining explicit consent may reduce the completeness of the data collection, which can introduce bias and reduce the representativeness of the results. Even if consent is granted, individuals must trust that their data will be used responsibly and not misused for other purposes.

For these reasons, we challenge the feasibility of robust privacy protection within a system that relies on identifying travelers to count them. Instead, we propose an alternative system that offers statistical counts of travelers as the only accessible information. To implement such a system, we propose to use Bloom filters, which are probabilistic data structures that support set operations, in combination with homomorphic encryption, which is a type of encryption that allows performing operations on encrypted data. We envision a system that provides reliable counts of travelers moving between locations as the only retrievable information [10].

In this paper, we explain and briefly evaluate our privacy-preserving method for its accuracy in counting travelers moving between locations, with the aim to show its principal working. As a case study, we consider a subway network where travelers utilize smart-card technology to check in and out of the transportation system. To accomplish this, we use a synthetic dataset that is accurate in representing the characteristics of a real-world subway dataset. The results demonstrate the effective combination of Bloom filters and homomorphic encryption in accurately counting travelers between locations while preserving individual privacy. This finding paves the way for expanding the analysis to include multiple locations within the subway network. Our research carries significant implications for enhancing public transportation efficiency and safeguarding user privacy.

2 System model

Our example proof-of-concept assumes a subway network with an automatic fare collection system. Subway networks usually consist of lines that connect specific origin and destination stations. For each station A, we assume there is a set of n_A scanners $\mathcal{S}_A = \{s_1^A, ..., s_{n_A}^A\}$, which are used by travelers to check in and out. In our model, we trust the sensors, but not the centralized server. For this reason, we first let a sensor collect detections to then send this collection in encrypted form to the server. The time during which detections are collected and aggregated before sending them to the server is called an **epoch**. Typically, an epoch lasts 5 minutes. As we will discuss in detail below, the server can operate on the encrypted collections of detections, but cannot reconstruct individual detections themselves.

A scanner $s \in \mathcal{S}_A$ reads a card's unique identifier *cid*. Each card reading belongs to an epoch $e \in \mathcal{E}$ corresponding to its timestamp t, such that $t_{start}(e) \leq t < t_{end}(e)$, where t_{start} and t_{end} mark the beginning and the end of an epoch and $\mathcal{E}$ denotes the set of all such epochs. A detection is thus a triplet (cid, s, e), representing a card uniquely identified by its identifier *cid*, read by scanner s during epoch e. By $\mathcal{D}_{s,e}$, we denote the set containing all the identifiers detected by a scanner s during an epoch e. Let $\mathcal{D}_e^A$ denote the set of all identifiers detected by *any* scanner at A during epoch e: $\mathcal{D}_e^A = \cup_{s \in \mathcal{S}_A} \mathcal{D}_{s,e}$.

Using collections of detections provides a powerful mechanism for counting travelers. One simple example is that the size of a set $\mathcal{D}_{s,e}$ indicates the number of travelers who passed the scanner s during the epoch e. More interestingly, for two different stations, the size of the set $\mathcal{D}^A_{e_1} \cap \mathcal{D}^B_{e_2}$ represents the number of travelers who were first detected at A during epoch e_1 and subsequently detected at B during epoch e_2, where e_2 occurs after e_1.

3 Method

3.1 Bloom filter

The problem with using sets of detections is that they still contain the card identifiers for anyone to see who has access to those sets. This issue can be addressed by using a *representation* for sets, called **Bloom filters** [3]. A Bloom filter has the property that it allows only for membership tests. In other words, the only way to discover which card identifier is stored, is to go over the entire list of possible card identifiers and check for each one of them which identifier the membership tests succeed. Although this already poses a potentially tremendous computational burden for discovering detected identifiers, it is not enough to prevent finding identifiers. To understand how encryption, combined with Bloom filters, can prevent such a discovery, we must first explain what they are.

A Bloom filter is implemented as a binary vector of m bits, initially all set to zero. Adding an element to the set involves hashing it with k different hash functions, each returning a position in the vector. Those bits are then set to 1. To determine whether an element is in the set, the same hash functions are applied, and the corresponding bits in the vector are checked. When each bit is also 1, the element is considered to be in the set. An important observation is that there is a chance that two different elements will see exactly the same bits being set to 1. As a consequence, a membership test may return a *false positive*: the element for which the test is computed is factually *not* in the set represented by the Bloom filter. It is for this reason that Bloom filters are said to be probabilistic data structures. Given the maximum acceptable probability p for false positives, along with the desired number n of elements to be stored, one can compute the minimal length m of a Bloom filter, as well as the minimal number k of hash functions to use: $m = -\frac{n \cdot \ln p}{(\ln 2)^2}$ and $k = \frac{m}{n} \cdot \ln 2$.

The size of the set represented by a Bloom filter (i.e., its *cardinality* c) can be estimated when knowing only k, m, and the number t of bits that are set to 1 [8]:

$$c = -\frac{m}{k} \ln\left(1 - \frac{t}{m}\right) \tag{1}$$

In addition to membership testing, Bloom filters also support union and intersection operations. An intersection of two sets $\mathcal{D}_A$ and $\mathcal{D}_B$ can be done by taking their respective Bloom filter representations and conducting a bitwise AND operation. To illustrate, if A is represented by $[0, 1, 1, 0, 1]$ and B by $[1, 1, 1, 0, 0]$, then $A \cap B$ is represented by $[0, 1, 1, 0, 0]$. A union is computed through a bitwise OR operation. (Note that for realistic representations of sets, Bloom filters generally have lengths of 1000s of bits.) Whereas unions do not affect the probability of false detections, intersections do. This also means that estimating the size of an intersection when using Bloom filters may easily see deviations. We ran extensive tests and encountered estimates that were 15% off the real size. A more accurate estimation for two intersecting sets is provided by [8], yet no general estimation is known for more than two intersecting sets. For this paper, we will use the simple approximation given by Equation 1.

Ignoring encryption for the moment, detections at a scanner s are converted into Bloom filters and sent by s to the server at the end of each epoch. To answer queries, the server may do a series of unions and intersections on various Bloom filters, as we explained in our simple

example above. The result is a Bloom filter representing the detections related to the query. At that point, the server could return the estimated cardinality of the set. Unfortunately, the server itself can still, with some computational effort, discover the detected card identifiers. As we mentioned, in our system model, we do not trust the server. This is where encryption comes into play.

3.2 Homomorphic Encryption

To prevent the server from discovering identifiers, Bloom filters must be combined with encryption schemes. Homomorphic encryption [9] is a specialized form of encryption that enables mathematical operations to be conducted directly on encrypted data without the need for decryption. The results of these operations are also encrypted, and the output is the same as if the operations had been conducted on unencrypted data.

The following procedure is now followed using homomorphic encryption. Suppose a user U is interested in knowing how many travelers moved from A to B. To that end, she passes an *encryption key* to the server, which is then used to encrypt all Bloom filters from the moment the key is available (note that this means that a user cannot issue queries that relate to the past, i.e., the time before they made the encryption key available). Also note that the user holds the *decryption key*, and is thus the only entity who can decrypt the corresponding encrypted Bloom filters. Neither the scanners nor the server can decrypt those Bloom filters.

The server now operates on bitwise encrypted Bloom filters and produces a final result, say an encrypted Bloom filter BF representing a set R. By simply *adding* the entries of BF, it can produce an (encrypted) version t^* of t, the number of bits that have been set to 1. This value, along with k and m can then be handed over to the user U, who can decrypt t^* and compute the cardinality c. The server can also hand out BF to the user, but not after having shuffled the entries (otherwise, the user could still decrypt BF and discover detections). Shuffling keeps the same number of (encrypted) bits that have been set to 1, but a shuffled version of BF has no relationship to R anymore.

4 Results and Discussion

In this section, to get a clear understanding of the effects of preserving privacy, we conduct an experiment by using a synthetic dataset. To determine the accuracy of the responses, we compare the statistical counts generated by our model with those from our dataset. For the hashing part, we choose MurmurHash3 [1], which is highly efficient. The estimation formula used by Bloom filters provides only an *approximation* of the number of elements likely to be present in the original set, rather than an exact count. For this reason alone, we expect to see deviations from the ground truth. In addition, taking intersections also affects the probability of having false positives; which will generally lead to overestimations of the size. We express the accuracy of the estimated count c to the real count c_t as:

$$Accuracy = \max\left(1 - \frac{|c - c_t|}{c_t}, 0\right) \tag{2}$$

To simulate real-world subway data, we generate card identifiers from a uniform distribution. As is common practice, real identifiers are often processed using a cryptographic hash function before being used for further analysis, and our use of uniform random identifiers similarly mimics this step. As an example, we ask ourselves how many travelers move from one station to another. Let $s_1^A, \ldots, s_{n_A}^A$ be the sensors at station A and $s_1^B, \ldots, s_{n_B}^B$ the sensors at station B, The answer is then $|\bigcup_{e_d} \bigcup_{e_a} \mathcal{D}_{e_d}^A \cap \mathcal{D}_{e_a}^B|$, where we assume that $e_d \triangleleft e_a$.

In other words, we take all combinations of departure epoch at A and *later* arrival epoch at B, and consider the detections from all sensors at A, and intersect that with the set of detections from all sensors at B.

To see the effects of taking intersections as unions, we count in two different ways. First, we simply compute the size of the union of intersections, as just mentioned. Second, we take a look at any combination of departure epoch e_d and (possible) arrival epoch e_a, as well as all pairs of sensors s_i^A and s_j^B. Using Bloom filter representations, we compute the size of the intersection $\mathcal{D}_{e_d}^A \cap \mathcal{D}_{e_a}^B$, and subsequently add those sizes for all combinations of departure and arrival epochs: $\sum_{e_d} \sum_{e_a} |\mathcal{D}_{e_d}^A \cap \mathcal{D}_{e_a}^B|$.

Table 1 Comparison of the accuracy of estimated counts with ground truth.

Ground truth	100	1000	10000	100000
Estimated count first method	96	1006	10001	99933
Accuracy first method	96.00%	99.40%	99.99%	99.93%
Estimated count second method	97	990	10081	107670
Accuracy second method	97.00%	99.00%	99.19%	92.33%

We conducted four experiments using $100, 1000, 10000$, and 100000 trips distributed over a single day (24h). For each experiment, we set the epoch length as 5 minutes (resulting in 288 epochs), fixed p at 0.001, and selected n to be equal to the corresponding number of trips in each experiment. We used optimal settings for m and k, given n and p. We ran each experiment 50 times. In both counting methods we perform a bitwise intersection with all possible arrival epochs for each departure epoch. The distinction between the counting methods takes into effect when performing intersections.

Table 1 presents the results of our experiments. The table displays the ground truth, as well as the estimated counts obtained using the two different counting methods from our proposed approach, along with the corresponding accuracy values. The results show that the difference between the counting methods becomes more pronounced as the number of trips increases. This is mainly because as epochs become more crowded, i.e., when we have more detections in a single epoch, the probability of false positives also increases when intersecting two epochs. The impact of false positives on the counting accuracy differs between the two methods. The first method yields an estimated count closer to the ground truth because it also considers the union of intersections. Taking the union ensures that false positives inside all intersections are counted only once because they are consolidated through the union operation at the end. In contrast, the second counting method estimates the size immediately after the intersection between each departure epoch at station A and each arrival epoch at station B. The estimated count after each intersection also includes false positives between the corresponding epochs. Therefore, the total count obtained at the end of this method is the sum of the counts obtained after each intersection, which includes false positives and leads to overestimating the total number of travelers. The first method's estimated count of travelers for all different numbers of trips is in close agreement with the actual count and consistently achieves high accuracy.

The difference in accuracies comes from the way we use Bloom filters, and is seen to be dependent on the query. Further research is needed to see how accuracies depend on different types of queries.

The current setup and implementation allow us to run queries involving (tens and hundreds of) thousands of travelers, often within just a few minutes, even with more intricate queries. When considering entire networks, many queries can be subdivided into independent parts, making them excellent candidates for processing in parallel.

5 Conclusion

In this paper, we have used a privacy-preserving method for counting travelers moving in public transport systems through encrypted Bloom filters. By using encrypted Bloom filters, we can count travelers moving between stations without revealing any information about who made these trips. Further, the information about who made which travels is unrecoverable and hidden for all components and parties in the system: the sensors, the server, and the client interested in the counts. The downside is that the method decreases the accuracy of counting. We evaluate the accuracy of our method on a synthetic subway dataset. We show that the loss of accuracy can be minimized and that it is possible to achieve highly accurate counting while ensuring that data cannot be used to trace back to an individual. An important observation is that the attainable accuracy is dependent on *how* counting takes place. If we count too soon to aggregate counts later on, we may fail to compensate for false counting later in the process. In other words, the accuracy of our method is dependent on the query and when counting and aggregation actually take place.

Although we did not show in this paper, our method is not limited to counting travelers moving between only two locations. The proposed method has the capability to handle more complex queries, such as counting the number of travelers moving between multiple locations. As a next step, we plan to investigate how more complex queries can also be answered with high accuracy. In addition, we need to investigate the practical feasibility of running queries such that answers can be provided in a reasonable time.

References

1 Austin Appleby. Murmurhash3.(2016). *URL: https://github. com/aappleby/smhasher-/wiki/MurmurHash3*, 2016.

2 Kay W Axhausen, Andrea Zimmermann, Stefan Schönfelder, Guido Rindsfüser, and Thomas Haupt. Observing the rhythms of daily life: A six-week travel diary. *Transportation*, 29(2):95–124, 2002.

3 Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

4 Yves-Alexandre De Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports*, 3(1):1–5, 2013.

5 Merkebe Getachew Demissie, Santi Phithakkitnukoon, Titipat Sukhvibul, Francisco Antunes, Rui Gomes, and Carlos Bento. Inferring passenger travel demand to improve urban mobility in developing countries using cell phone data: a case study of senegal. *IEEE Transactions on intelligent transportation systems*, 17(9):2466–2478, 2016.

6 Yola Georgiadou, Rolf A de By, and Ourania Kounadi. Location privacy in the wake of the gdpr. *ISPRS International Journal of Geo-Information*, 8(3):157, 2019.

7 Dmytro Karamshuk, Chiara Boldrini, Marco Conti, and Andrea Passarella. Human mobility models for opportunistic networks. *IEEE Communications Magazine*, 49(12):157–165, 2011.

8 Odysseas Papapetrou, Wolf Siberski, and Wolfgang Nejdl. Cardinality estimation and dynamic length adaptation for bloom filters. *Distributed and Parallel Databases*, 28:119–156, 2010.

9 Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

10 Valeriu-Daniel Stanciu, Maarten van Steen, Ciprian Dobre, and Andreas Peter. Privacy-preserving crowd-monitoring using bloom filters and homomorphic encryption. In *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, pages 37–42, 2021.