# Finding Persistent Items in Data Streams

Haipeng Dai[1] Muhammad Shahzad[2] Alex X. Liu[1] Yuankun Zhong[1]
[1]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu, CHINA
[2]Department of Computer Science, North Carolina State University, Raleigh, NC, USA
haipengdai@nju.edu.cn, mshahza@ncsu.edu, alexliu@cse.msu.edu, kun@smail.nju.edu.cn

## ABSTRACT

Frequent item mining, which deals with finding items that occur frequently in a given data stream over a period of time, is one of the heavily studied problems in data stream mining. A generalized version of frequent item mining is the persistent item mining, where a persistent item, unlike a frequent item, does not necessarily occur more frequently compared to other items over a short period of time, rather *persists* and occurs more frequently over a long period of time. To the best of our knowledge, there is no prior work on mining persistent items in a data stream. In this paper, we address the fundamental problem of finding persistent items in a given data stream during a given period of time at any given observation point. We propose a novel scheme, PIE, that can accurately identify each persistent item with a probability greater than any desired false negative rate (FNR) while using a very small amount of memory. The key idea of PIE is that it uses Raptor codes to encode the ID of each item that appears at the observation point during a measurement period and stores only a few bits of the encoded ID in the memory of that observation point during that measurement period. The item that is persistent occurs in enough measurement periods that enough encoded bits for the ID can be retrieved from the observation point to decode them correctly and get the ID of the persistent item. We implemented and extensively evaluated PIE using three real network traffic traces and compared its performance with two prior adapted schemes. Our results show that not only PIE achieves the desired FNR in every scenario, its FNR, on average, is 19.5 times smaller than the FNR of the best adapted prior art.

## 1. INTRODUCTION

### 1.1 Motivation and Problem Statement

With the increase in the popularity of applications that process large volumes of data generated at high rates, such as internet traffic analysis [12], business decision support [3], direct marketing [20], and sensor data mining [2], data stream mining is becoming more and more important. Data stream mining is the process of extracting information from a data stream, where a data stream is an ordered sequence of data items that can often only be read once using limited computing and storage resources [11]. One of the heavily studied problems in data stream mining is the fundamental problem of mining frequent items, which deals with finding items that occur most frequently in a given data stream over a period of time [7, 8, 17–19]. Frequent item mining finds applications in a variety of practical scenarios such as finding most popular destinations or heaviest users on the Internet by treating packets as items [7], and the most popular terms in queries made to an Internet search engine [22].

A more generalized version of frequent item mining is the *persistent frequent item* mining. A persistent frequent item, unlike a frequent item, does not necessarily occur more frequently compared to other items over a short period of time, rather *persists* and occurs more frequently compared to other items only over a long period of time. For the sake of brevity, onwards, we will call persistent frequent items just persistent items. Persistent item mining finds applications in a variety of settings such as network security and click-fraud detection. For network security, persistent item mining can be used to detect stealthy DDoS attacks, where an attacker does not overwhelm the target rather degrades its performance using a small number of attacking machines for a long period of time [26]. Similarly, it can be used to detect stealthy port scanning attacks, where an attacker uses a small probing rate for a long period of time to discover system vulnerabilities [26]. Persistent item mining can also be used to detect network bots by monitoring the communication between a bot and its C&C server [13]. For click-fraud detection, persistent item mining can be used to detect if automatic robots are periodically generating clicks on an ad to increase the payment for an advertiser in pay-per-click online advertising systems [15].

In this paper, we address the fundamental problem of finding, or in other words, *identifying* persistent items in a given data stream during a given period of time at any given observation point. An *observation point* is any computing device that can see the data stream and can process and store information about items in the data stream. Examples of observation points include end hosts, network middleboxes (such as routers and switches), firewalls, and other computing devices. To formally state the problem statement, let us divide the given period of time into small equally sized measurement periods. Let us also define the term *occurrence* as an event that one or more instances of an item appear in a

*Proceedings of the VLDB Endowment,* Vol. 10, No. 4
Copyright 2016 VLDB Endowment 2150-8097/16/12.

measurement period. Formally, *given a period of time comprised of $T$ consecutive equally sized measurement periods, a threshold $T_{th}$, and a desired false negative rate, identify the persistent items,* i.e., *the items that occur in more than $T_{th}$ out of the $T$ measurement periods, such that the percentage of persistent item not successfully identified is less than the desired false negative rate.*

## 1.2 Limitations of Prior Art

To the best of our knowledge, there is no prior work on identifying persistent items in a data stream. The most related problem is the extensively studied frequent item mining problem [7, 8, 18, 19]. Unfortunately, none of the existing schemes for mining frequent items can be directly used for mining persistent items because frequent item mining schemes count the frequencies of items without taking the temporal dimension into account. More specifically, a frequent item mining scheme can calculate the total number of instances in $t$ measurement periods but it can not calculate the number of measurement periods in which those instances occurred out of the $t$ measurement periods. For example, in a stealthy DDoS attack, if an attacking machine sends $10,000$ packets during 100 measurement periods while a legitimate machine sends the same number of packets in 2 measurement periods, existing frequent item mining schemes can calculate that both machines sent $10,000$ packets each, but can not determine the amount of time they took to send these packets. Furthermore, as existing schemes for mining frequent items do not take the temporal dimension into account, they cannot eliminate duplicate items within a measurement period, such as multiple packets belonging to the same flow, which can result in the overcounting of the number of occurrences of an item.

## 1.3 Proposed Approach

In this paper, we propose PIE, a Persistent items Identification schemE. PIE can accurately identify each persistent item with a failure probability lower than the desired false negative rate (FNR), where FNR is defined as the ratio of the number of persistent items that PIE fails to identify to the number of all persistent items. The key idea of PIE is that it uses Raptor codes (proposed in [24]) to encode the ID of each item that appears at a given observation point during a measurement period and stores only a few bits of the encoded ID in the memory of that observation point during that measurement period. The ID of an item can be any arbitrary identifier such as for network IP packets, it can be the standard five tuple (i.e., source IP, destination IP, source port, destination port, and protocol type). The motivation behind encoding the ID with Raptor codes and then storing the result instead of storing the original ID is that the Raptor codes encode the given ID into a potentially limitless sequence of bits. To decode the bits to obtain the original ID, Raptor codes need only a small subset of all the encoded bits. As we shall see later, it is possible that when an observation point tries to store information (either encoded or non-encoded) about ID of an item at a memory location, that memory location may already be occupied, resulting in a collision and loss of ID information. Had we been storing original non-encoded ID, the bits that the observation point could not store due to collision would be lost and the original ID may never be retrieved correctly. On the contrary, when storing bits for the ID of an item encoded with Raptor codes, even if some bits are lost due to collision during a few measurement periods, if the item is persistent, there will still be enough encoded bits stored during other measurement periods such that they can be correctly decoded to get the item ID. The motivation behind storing only a few bits of the encoded ID during each measurement period is to minimize the memory required at the observation point. The item that is persistent will occur in enough measurement periods that enough encoded bits for the ID can be retrieved from the observation point to decode them correctly and get the item ID. If we store enough encoded bits for each item during each measurement period such that the bits could be decoded to obtain the original ID from just a single measurement period, it would only result in wasted memory. To identify persistent items, PIE simultaneously processes the information of IDs stored during all measurement periods. For each persistent item, with a high probability, PIE obtains enough encoded bits to decode them correctly and get the ID of the persistent item. We have theoretically calculated the expression for FNR. PIE uses this expression to calculate the optimal number of encoded bits for each ID that it should store during the given measurement period such that the actual FNR never exceeds the desired FNR and at the same time, the amount of memory consumed at the observation point in storing the encoded bits is minimum.

## 1.4 Key Technical Challenges

The first technical challenge is to store the IDs of persistent items without requiring large amounts of memory at the observation point. To address this challenge, instead of storing original IDs of items during each measurement period, we use Raptor codes to encode the ID of each item and then store only a few bits of the encoded ID during each measurement period. To ensure that the original ID of each persistent item can be recovered with a high probability while consuming minimum amount of memory, we theoretically calculate the expression for FNR as a function of the length of Raptor codes, and then determine the minimum required number of encoded bits to be stored during each measurement period.

The second technical challenge is to accurately recover IDs of items using the encoded bits from multiple measurement periods. The encoded bits for the ID of an item in a measurement period are different from the encoded bits for the ID of that item in a different measurement period. Therefore, PIE cannot determine which sets of encoded bits belong to the same ID. To address this challenge, PIE calculates a hash-print of each ID by applying a hash function to the ID and stores it along with the encoded bits during each measurement period in which the item with that ID appears. Unlike the encoded bits, the hash-print stays the same across measurement periods. PIE uses this hash-print to first group all sets of encoded bits across the $T$ measurement periods that have the same hash-print and then recovers the ID using only the encoded bits in this group. It is possible that the hash-print of two or more different IDs may be the same. To address this challenge, after recovering an ID from a group, PIE applies a two-step verification and accepts the recovered ID only of the ID passes the two tests.

## 1.5 Key Contributions

PIE brings forward the state of the art in persistent item identification on the following three fronts: reliability, scalability, and robustness. For reliability, PIE takes the desired

FNR as input from the system operator and ensures that the actual FNR is smaller than the desired FNR. For scalability, PIE stores information about all items in a single data structure during one measurement period regardless of the number of different items and the number of times each item appears at the observation point. For robustness, PIE is robust to accidental events, such as loss of information due to SRAM failures, because the use of Raptor code based encoding makes PIE inherently robust to losing some encoded bits, and still be able to decode the available bits to correctly recover IDs.

We extensively evaluated PIE and compared it with two prior schemes IBF [9,14] and CM sketch [8] that we adapted to enable persistent item identification. In our evaluations, we used three real-world network traffic traces including a backbone traffic trace [1], an enterprise network traffic trace [21], and a data center traffic trace [4]. Our results also show that the actual FNR of PIE is always less than the desired FNR. Our results further show that PIE achieves, on average, 19.5 times lower FNR compared to the adapted IBF scheme. Similarly, our results show that PIE achieves at least 426.1 times lower FPR compared to the adapted CM sketch.

## 2. RELATED WORK

To the best of our knowledge, there is no prior work on identifying persistent items in a data stream. However, a large body of works has been devoted to identifying frequent items, and some of the existing schemes can be adapted to identify persistent items. The frequent items problem (also known as the heavy hitters problem) refers to finding the items that occur most frequently in a given data stream of items. There are two main classes of algorithms for finding frequent items: counter-based algorithms and sketch-based algorithms [7]. Next, we first briefly describe the prior work on finding frequent items belonging to these two classes. After that, we describe an orthogonal class of work, namely Invertible Bloom Filter (IBF), that is not designed for finding frequent items, but can be modified to identify persistent items.

**Counter-based Algorithms:** Manku *et al.* proposed the Lossy Counting (LC) algorithm to find frequent items [18]. The LC algorithm maintains a tuple of a lower bound on the count and the difference between the upper bound and the lower bound for each item. For each item, LC either increments its count by 1 if the item was previously seen, or adds it into the memory if the item is new and appropriately sets the lower bound for it. LC algorithm does not have any false negatives and its false positives are bounded, where the bound is proportional to the size of data stream. Metwally *et al.* proposed the Space Saving (SS) algorithm, which stores $k$ (item, count) pairs [19]. If the incoming item matches an item in one of the $k$ pairs, SS increments the count in the corresponding pair by one. Otherwise, it replaces the pair with the smallest count value by a new pair, which contains the incoming item and the count equal to the smallest count value incremented by one. Like LC algorithm, SS algorithm also does not have any false negatives and its false positives are bounded, where the bound is proportional to the size of data stream. These two algorithms and other such counter-based algorithms need to keep a set of items and counters. Thus, their errors grow proportionally with the size of the data stream, which is unacceptable for streams with a large number of frequent items or with huge size.

**Sketch-based Algorithms:** Sketch-based algorithms use sketches, which essentially contain linear projections of the input. Charikar *et al.* proposed Count (C) sketch [6] that consists of a two-dimensional array of counters. For each input item, C-sketch maps it to a counter in each row of counters using a hash function. In each mapped counter, C-sketch adds a value $\in \{-1, +1\}$ using another hash function. The total space, time per update, and error are determined by the width and height of the array. Cormode *et al.* proposed the famous Count-Min (CM) sketch that also consists of a two-dimensional array of counters [8]. For each input item, CM-sketch maps it to a counter in each row of counters using a hash function and increments each of the mapped counter by 1. To estimate the number of times any given item appeared, CM-sketch first uses the hash-function to identify the counters in each row that the item maps to, then returns the value of the smallest counter as the estimate. CM-sketch consumes less space than C-sketch for a given error requirement. Both C- and CM-sketch can be extended to identify persistent items by allocating enough memory to store information about the ID of all items. Unfortunately, the amount of memory that these two algorithms require becomes prohibitively large when the number of items in a data stream is large. A common short-coming of both counter-based and sketch-based algorithms is that as they do not take the temporal dimension into account, they cannot eliminate duplicate items within a measurement period, such as multiple packets belonging to the same flow, which can result in the over-counting of the number of occurrences of an item for our persistent item problem. For this, one can maintain a conventional Bloom filter as described earlier in Section 6.1.

**Invertible Bloom Filter:** An invertible Bloom filter (IBF) is similar to a conventional Bloom filter except that it can be inverted to yield the IDs of some of the inserted items [9,14]. An IBF contains an array of cells, which in turn contains three fields: idSum, hashSum, and count. For any incoming item, the item is mapped into several cells using hash functions. For each cell that the item is mapped to, the count is incremented by 1, the stored idSum is XORed with the item ID, and the stored hashSum is XORed with the hash of the item ID. In the decoding phase, the "pure" cells, *i.e.*, the cells with count field either 1 or $-1$, are first identified and the item IDs from them are recovered. Using the IDs recovered from these pure cells, IBF further decodes the IDs from other cells to which the recovered IDs were mapped to. Note that, IBF can not always decode all IDs, *i.e.*, it has false negatives. IBF can be adapted for identifying persistent items by assigning one IBF to each measurement period and maintaining an additional Bloom filter as described earlier in Section 6.1. Unfortunately, this adapted IBF scheme has a very low memory efficiency because it needs to store the whole ID information for every item during every measurement period, unlike PIE, which stores only a few encoded bits of every ID during any given measurement period in which the item appeared.

## 3. PIE – ALGORITHM

PIE has two phases: recording phase and identification phase. In the recording phase, PIE records information about the ID of each item seen at the observation point during the given measurement period. To store this information, during each measurement period, PIE maintains a data structure called Space-Time Bloom Filter (STBF) in SRAM and records information about the items in this data structure. At the end of each measurement period, PIE transfers this data structure to the permanent storage for later analysis and starts a new instance of STBF in the next measurement periods. At the end of $T$ measurement periods, PIE has $T$ instances of STBF stored in its memory. In the identification phase, PIE analyzes these $T$ instances of STBF to identify persistent items during the $T$ measurement periods. Next, we first describe STBF and then explain PIE's recording and identification phases.

Due to space constraints, we omit the table of notations used in this paper.

## 3.1 Space-Time Bloom Filter

STBF can be regarded as an extended version of the conventional Bloom filter (BF) [5], which not only records the membership information of items in a set but also records information about their IDs. Let $h_1(.), \cdots, h_k(.)$ be $k$ independent hash functions with uniformly distributed outputs. Given a set of elements $S$, BF first constructs an array $A$ of $m$ bits, where each bit is initialized to 0, and for each item $e$ in $S$, BF sets the $k$ bits $A[h_1(e)\%m], \cdots, A[h_k(e)\%m]$ to 1. To process a membership query of whether item $e$ is in set $S$, BF returns true if all corresponding $k$ bits are 1 (*i.e.*, returns $\wedge_{y=1}^{k} A[h_y(e)\%m]$). In STBF, each bit in BF is replaced by a *cell*. Thus, STBF is an array $C_i$ of $m$ cells. For each item $e$ that appears during a measurement period $i$, PIE applies the same $k$ independent hash functions $h_1(.), \cdots, h_k(.)$ on $e$ to identify the cells $C_i[h_1(e)\%m], \cdots, C_i[h_k(e)\%m]$ in the array $C_i$ of the STBF in this measurement period to which this item $e$ maps. Each cell is comprised of three data fields: flag field, Raptor code field, and hash-print field. The Raptor code field of cell $x$, where $1 \leq x \leq m$, stores an $r$-bit Raptor code of the ID of an item $e$ if some hash function maps the item $e$ to this cell, *i.e.*, if for some $y$, $h_y(e)\%m = x$, where $1 \leq y \leq k$. Typically $r$ is much smaller than the length of item IDs. PIE selects the value of $r$ such that the probability of decoding the IDs of persistent flows is high during the identification phase, while at the same time the amount of memory Raptor code fields consume in the STBF is minimum. We will present the mathematical model that PIE uses to calculate the optimal value of $r$ in Section 5.2. PIE uses the information stored in the Raptor code fields of cells to recover the exact IDs of persistent items during the identification phase. We represent the Raptor code field of cell $x$ with $C_{iR}[x]$. The hash-print field of cell $x$ stores an $p$-bit hash-print of the ID of an item $e$ if some hash function maps the item $e$ to this cell, *i.e.*, if for some $y$, $h_y(e)\%m = x$, where $1 \leq y \leq k$. The hash-print of an item ID is calculated using a hash function, which is different from the $k$ hash functions used to map the item to $k$ cells in STBFs. PIE uses the hash-print values stored in the hash-print fields of cells during the identification phase to group the cells across different STBFs that are storing the Raptor codes for the same ID. We represent the hash-print field of cell $x$ with $C_{iP}[x]$. PIE selects the value of $p$ such that the probability of collision between hash-prints of different IDs is small, while at the same time the amount of memory hash-print fields consume in the STBF is minimum. We will present the mathematical model that PIE uses to calculate the optimal value of $p$ in Section 5.2. The flag field is just a single bit that indicates whether the cell is empty, singleton, or collided depending on whether the hash functions of none, one, or more than one items, respectively, mapped them to this cell. We represent the flag field of cell $x$ with $C_{iF}[x]$. More specifically, $C_{iF}[x] = 1$ indicates that only one item is mapped to cell $x$, *i.e.*, cell $x$ is a singleton cell. If $C_{iF}[x] = 0$ and $C_{iR}[x] = C_{iP}[x] = 0$, this indicates that no item is mapped to cell $i$, *i.e.*, cell $x$ is an empty cell. If $C_{iF}[x] = 0$ and $C_{iR}[x] = 2^r - 1$ and $C_{iP}[x] = 2^p - 1$, *i.e.*, all bits of Raptor code field and hash-print field are set to 1, this indicates that more than one item are mapped to cell $x$, *i.e.*, cell $x$ is a collided cell.

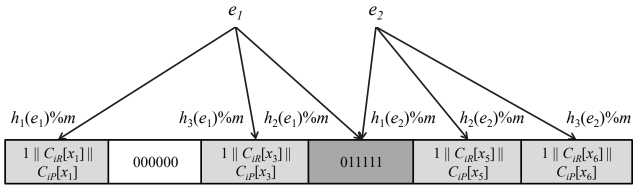

$e_1$
$e_2$
$h_1(e_1)\%m$
$h_3(e_1)\%m$
$h_2(e_1)\%m$
$h_1(e_2)\%m$
$h_2(e_2)\%m$
$h_3(e_2)\%m$
$1 \| C_{iR}[x_1] \| C_{iP}[x_1]$
000000
$1 \| C_{iR}[x_3] \| C_{iP}[x_3]$
011111
$1 \| C_{iR}[x_5] \| C_{iP}[x_5]$
$1 \| C_{iR}[x_6] \| C_{iP}[x_6]$

**Figure 1: Space-time code Bloom filter**

Figure 1 shows a simple toy example of STBF. The illustrated STBF has 6 cells, and applies 3 different hash functions to each item. For the two distinct items $e_1$ and $e_2$, STBF maps them to three cells each. There is one cell, the fourth from left, which both items are mapped to and is thus collided. There is one cell, the second from the left, which no item mapped to and is thus an empty cell with all bits set to 0. All remaining cells have the flag fields set to 1, and the appropriate Raptor codes and hash values stored.

Note that we chose to extend BF to build STBF. Another data structure that provides similar functionality but better performance compared to BF is Cuckoo filter [10]. Next, we explain why we did not choose to extend Cuckoo filter to build STBF. Cuckoo filter maintains an array of buckets and maps each item to two buckets using two independent hash functions. For any given item, if either of the two buckets the item maps to is empty, Cuckoo filter inserts the item in that bucket. If neither of the two buckets is empty, then Cuckoo filter kicks out the existing item from one of the two buckets, inserts the new item into this vacated bucket, and tries to re-insert the kicked-out item to the other bucket it maps to. Cuckoo filter repeats this process until either a maximum number of kicks are made or the kicked out item finds an empty bucket. Unfortunately, these kicks and reinsertions can take a long time and still result in insertion failures. If we were to use Cuckoo filters, whenever insertion of a given packet takes a larger duration of time due to these kicks, some of the packets arriving immediately after the given packet cannot be processed in time and their information will not be recorded by Cuckoo filter. This indeed happens in reality because the inter-arrival time of packets is comparable with the processing time of recording [16]. As the insertion time of Cuckoo filter is randomly distributed, any packet has similar probability to be missed. Thus, persistent items, which already have very few packets in each measurement period, will also be missed. In contrast, traditional BF has deterministic insertion time and can, therefore, be implemented to process packets at line-rate. As a result, we chose to extend BF to build STBF instead of extending Cuckoo filter.

## 3.2 Recording Phase

In the recording phase, PIE records information about the ID of each item seen at the observation point during the given measurement period. To store this information, at the start of each measurement period, PIE initializes STBF in SRAM, records information about the items in it, and dumps it into the permanent storage at the end of the measurement period. For STBF initialization, PIE sets all three fields (flag, Raptor code, and hash) in all cells of the STBF to 0. For each item $e$ seen at the observation point in the measurement period $i$, PIE performs the following three steps. First, it computes an $r$-bit Raptor code of the ID of the item and a $p$-bit hash-print of the ID. Second, it calculates the $k$ hash functions $h_y(e)$, where $1 \leq y \leq k$ and identifies the $k$ cells $C_i[h_y(e)\%m]$ to which the element maps. Third, for each cell $C_i[h_y(e)\%m]$, PIE checks whether the cell is empty, singleton, or collided. **If the cell is empty**, PIE sets the flag field $C_{iF}[h_y(e)\%m]$ to 1, inserts the Raptor code of the ID in $C_{iR}[h_y(e)\%m]$, and the hash-print of the ID in $C_{iP}[h_y(e)\%m]$. **If the cell is singleton**, PIE checks whether the Raptor code and the hash-print stored in the cell match the Raptor code and hash-print of the current item $e$. If they do, there is a high probability, that item $e$ was seen earlier as well during the current measurement period, and PIE had set the Raptor code and hash-print fields of this element at that time. In this case, PIE does not need to store the information of the current item $e$ again because it has already been stored in the cell when element $e$ was seen earlier during the current measurement period. If, however, the cell is singleton but the Raptor codes and hash-print previously stored in the cell do not match those of the current item $e$, then this singleton cell now becomes collided and PIE sets the flag field to 0 and all the remaining bits in the cell to 1, *i.e.*, $C_{iF}[h_y(e)\%m] = 0$, $C_{iR}[h_y(e)\%m] = 2^r - 1$, and $C_{iP}[h_y(e)\%m] = 2^p - 1$. **If the cell is collided**, PIE does not need to do anything further to process this cell.

### *3.2.1 Raptor Code Based Encoding*

Raptor code is a forward error correction (FEC) technology that is primarily used to provide application-layer protection against network packet loss [24]. The main advantages of Raptor codes are linear time encoding and decoding, the ability to encode a number of symbols into a potentially limitless sequence of symbols, and quite small decoding failure probability, $P_{df}$, which is the probability that one can not decode the symbols encoded through Raptor codes to obtain the original set of symbols. Let the length of the item ID be $l$ bits, and let we use $r$ bits to encode the ID using Raptor codes, $P_{df}$ is given by the following equation (derived in [24]).

$$P_{df}(r;l) = \begin{cases} 1, & \text{if} \quad r < l \\ 0.85 \times 0.567^{r-l}, & \text{if} \quad r \geq l \end{cases}. \qquad (1)$$

In this paper, we use traditional Raptor code, which operates over Galois field $GF(2)$. There is an enhanced version of Raptor code called RaptorQ code [25], which operates over $GF(256)$ and achieves better performance. Unfortunately, RaptorQ code is not suitable for our setting for two reasons. First, it is space inefficient. Second, its output has a byte level granularity, while the output of $GF(2)$ Raptor code has bit-level granularity. Typically the length of item IDs is no more than a few bytes. Thus, the required length of Raptor codes in each measurement period is just a few bits, which makes, $GF(2)$ Raptor code more suitable than RaptorQ code. Onwards, whenever we say Raptor code, we mean $GF(2)$ Raptor code.

The generalized encoding process based on Raptor codes is complicated, but in our setting, it can be viewed as a weighted linear sum of the bits in the item ID. Let us represent the ID of an element $e$ with a vector $\mathbf{I}^e = (I_1^e \ I_2^e \ ... \ I_l^e)$, where $I_x^e$ represents the $x^{\text{th}}$ bit in the ID $\mathbf{I}^e$. To encode the IDs of items during any measurement period $i$, at the start of the period, PIE uses $i$ as a seed to generate $r$ encoding-coefficient vectors $\mathbf{a}_{ij}$, where $1 \leq j \leq r$. Each encoding-coefficient vector $\mathbf{a}_{ij}$ contains $l$ encoding-coefficients $a_{ij}^x$, where $1 \leq x \leq l$. The encoding-coefficients $a_{ij}^x$ serves as the weight for the $x^{\text{th}}$ bit in the weighted linear sum of the bits of the item ID. Note that PIE uses same $r$ encoding-coefficients vectors for each item in a given measurement period, but the encoding-coefficient vectors for different measurement periods are different.

In a measurement period $i$, to generate the $j^{\text{th}}$ of the $r$ encoded bits of item ID $e$ represented by $R_{ij}^e$, which will be stored in the Raptor code field of appropriate cells, *i.e.*, in $C_{iR}[h_y(e)\%m]$, where $1 \leq y \leq k$, PIE takes the dot product of the encoding-coefficient vector $\mathbf{a}_{ij}$ with the ID vector $\mathbf{I}^e$. PIE obtains this dot product for all values of $j$, where $1 \leq j \leq r$, to get the $r$ bits of the Raptor code of the given item ID. This process is represented by the following equation.

$$\begin{pmatrix} a_{i1}^1 & a_{i1}^2 & \dots & a_{i1}^l \\ a_{i2}^1 & a_{i2}^2 & \dots & a_{i2}^l \\ \vdots & \vdots & & \vdots \\ a_{ir}^1 & a_{ir}^2 & \dots & a_{ir}^l \end{pmatrix} \cdot \begin{pmatrix} I_1^e \\ I_2^e \\ \vdots \\ I_l^e \end{pmatrix} = \begin{pmatrix} R_{i1}^e \\ R_{i2}^e \\ \vdots \\ R_{ir}^e \end{pmatrix} \qquad (2)$$

Note that PIE needs the IDs of all items to be of equal length, which is trivial because IDs with smaller length can be made equal in length to other IDs by appending appropriate number of 0s at the start of the ID.

### *3.2.2 Hash-print Calculation*

The process of calculating the hash-print is straight forward. PIE simply applies a hash function with uniformly distributed output to calculate the $p$-bit hash-print of any given ID and stores it in the hash-print field of the $k$ appropriate cells. Note that, while Raptor codes of a given ID are different across measurement periods, *i.e.*, $R_{ij}^e$ is different for different values of $i$, the hash-print value stays the same.

## 3.3 Identification Phase

In the identification phase, to recover the IDs of the persistent items with more than $T_{th}$ occurrences in any set of $T$ measurement periods, PIE processes the STBFs generated during those $T$ periods and recovers the IDs of all such persistent items with high probability. In Section 4.1, we will calculate the probability that PIE fails to recover the ID of any given persistent item.

To recover the IDs, PIE picks all the cells at the same cell position from the $T$ STBFs, which we name a *cell line*. Formally, all cells $C_1[x]$, $C_2[x]$, $\cdots$, $C_T[x]$ constitute a cell line, where $1 \leq x \leq m$. Note that each cell line is comprised of $T$ cells, and there are $m$ such cell lines. Figure 2 shows 6 STBFs obtained in 6 measurement periods. The highlighted group of cells at the 7th position constitute the cell line with $x = 7$. PIE starts with $x = 1$ and processes this cell line before moving to the cell line with $x = 2$ and so on until it has processed the last cell line with $x = m$, at which point PIE should have recovered IDs of all items that occurred

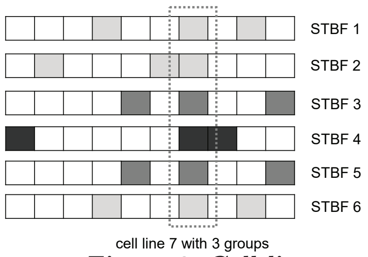


**Figure 2: Cell line**

in more than $T_{th}$ measurement periods. Next, we explain processing of each cell line.

### *3.3.1 Cell Line Processing*

To process the cell at location $x$ from the left, PIE performs the following three steps. First, it discards all empty and collided cells in this cell line because such cells do not contain information about any item. Second, out of the remaining cells in the cell line, PIE divides the cells into groups such that the hash-prints of all cells in the same group are the same. The motivation behind this type of grouping is that the cells containing the same hash-print are highly likely to contain the encoded bits of the ID of the same item because probability that hash-prints of two IDs will be the same is very small (but not zero; and we will shortly handle the case if hash-prints of multiple IDs indeed are the same). PIE uses the bits in the Raptor code fields of the cells in each group to recover the ID from that group. In our example in Figure 2, each item is mapped to 3 locations, *i.e.*, $k = 3$. To keep the example simple, we have assumed that in this particular example, each STBF contains information about only one item. The same shade of grey in this figure represents the information with same hash-prints (but may be from different items). In this example, cell line with $x = 7$ has 3 different groups marked by 3 different shades from 4 items. Note that the ID of the item in STBF 2 differs from that in STBF 1 and 6 as their three cell locations are not exactly the same. This leads to mingled groups as will be described in Section 3.3.3. Third, PIE compares the hash-prints of the groups generated from the current cell line with the hash-prints of the groups generated from the previous $x - 1$ cell lines, and merges the groups with the same hash-prints. PIE performs these three steps on all cell lines from $x = 2$ to $x = m$, except on the cell line with $x = 1$, on which it only performs the first two steps. Note that after processing all the cell lines, each group of cells has a unique hash-print.

### *3.3.2 Recovering Item IDs*

To recover the ID of the item from any given group containing $g$ cells, PIE uses Equation (2). PIE regenerates the encoding-coefficient vectors $\mathbf{a}_{ij}$, where $1 \leq j \leq r$ using the index $i$ of the measurement period, which it already knows. It observes the values of $R^e_{ij}$ from the cells in the group. It then plugs in all the values of $\mathbf{a}_{ij}$ and $R^e_{ij}$ into Equation (2) and obtains a set of $g \times r$ linear equations, which it solves to get the ID vector $\mathbf{I}^e$. Finally, it verifies that the recovered ID is correct. For this, it performs two verification tests. First, it calculates the hash-print of the recovered ID and compares it with the hash-prints in the cells of the group. If the hash-prints match, the recovered ID has passed the first test. Second, it applies the $k$ hash functions and checks the cells at locations $h_1(ID), \cdots, h_k(ID)$ in all those STBFs from which the cells in the group were taken. For the recovered ID to pass the second test, unless the cells at these $k$ locations in these STBFs are collided, the Raptor code fields of all these cells should be the same and the hash-prints of all these cells must match the hash-print of the recovered ID. If that is indeed the case, the recovered ID has passed the second test, and can be considered correct with high probability.

Note that the value of $r$ is much smaller than the ID length $l$ and to solve the linear equations successfully to recover $\mathbf{I}^e$, PIE needs at least $l$ equations, *i.e.*, PIE needs $g \times r$ to be $\geq l$, or in other words, it needs $g \geq \lceil l/r \rceil$ to be able to recover the original ID of the item of that group. PIE calculates the values of $r$ and $m$ such that the number of cells $g$ in the group of a persistent item, *i.e.*, the item with more than $T_{th}$ occurrences in $T$ periods, is greater than $\lceil l/r \rceil$ with a very high probability. We will present the mathematical equations that PIE uses to calculate the values of $r$ and $m$ in Section 5.2.

### *3.3.3 Mingled Groups*

Next, we describe how PIE recovers the IDs if the cells in a group contain encoded bits coming from two or more IDs instead of a single ID. We call such groups *mingled groups*. This happens when the hash-prints of two or more IDs are the same, *i.e.*, there is a hash-print collision. For instance, cell line 7 in Figure 2 indeed has 4 groups, but two groups among them are mingled groups. In such a setting, when PIE recovers an ID, the hash-print of the recovered ID will not match the hash-print stored in the hash-print fields of the cells in the group. In this case, PIE temporarily removes $g_r$ cells from the group and tries to recover the ID using only the remaining $g - g_r$ cells. More specifically, PIE starts with $g_r = 1$, and performs $\binom{g}{g_r}$ iterations, where in each iteration, it temporarily removes a unique set of $g_r$ cells from the original set of $g$ cells and recovers an ID using only $g - g_r$ cells. As soon as PIE recovers an ID whose hash-print matches that of the cells in the group, it records that ID and continues to perform the iterations to recover any more IDs whose cells might be present in this group. PIE increments $g_r$ by one when it has performed $\binom{g}{g_r}$ iterations using the current value of $g_r$, until the value of $g_r$ becomes equal to a threshold $g_T$. Although PIE can set the threshold to be equal to $g$, it typically does not do that because the value of $g$ can be quite large, resulting in a prohibitively large number of iterations. PIE typically sets the threshold $g_T$ to be no more than 2. We have observed from our extensive evaluation that PIE recovers, on average, 93.5% of persistent items using this threshold.

## 4. PIE – ANALYSIS

In this section, we derive the false negative rate (FNR) and false positive rate (FPR) of PIE.

## 4.1 False Negative Rate

There are two reasons that can cause PIE to fail to recover the IDs of persistent items: hash-mapping collisions and hash-print collisions. Next we describe these two reasons.

**Hash-mapping Collisions:** It is possible that several cells out of the $k$ cells $C_i[h_1(e)\%m], \cdots, C_i[h_k(e)\%m]$ to which a given item $e$ is mapped using the $k$ hash functions $h_1(e), \cdots, h_k(e)$ are collided because the $k$ hash functions may map other items to these cells, especially if the number of items is large. Due to such hash-mapping collisions, the number of cells $g$ in the group of item $e$ during the identification phase may not satisfy the requirement $g \geq \lceil l/r \rceil$, leading to the failure of PIE in recovering the ID of the item $e$.

**Hash-print Collisions:** It is possible that due to the limited number of bits in the hash-print fields of cells, the hash-prints of some items may be the same. Due to such hash-print collisions, PIE may put the cells of different items into the same group during the identification phase, *i.e.*, the group becomes *mingled*. Although PIE can recover IDs of multiple items from mingled groups as explained in Section 3.3.3, if the number of mingled cells is greater than the threshold $g_T$, PIE may fail to recover the IDs of any item from the group.

Next, we first derive the probabilities that PIE can recover the ID of any given item despite hash-mapping collisions and hash-print collisions. We name these probabilities "hash-mapping collision survival probability" $P_m$ and "hash-print collision survival probability" $P_p$. After that, we use $P_m$ and $P_p$ to derive an expression for the overall FNR of PIE.

### 4.1.1 Hash-mapping Collision Survival Probability

Let the total number of occurrences of all items during $T$ measurement periods be $N$. Thus, the expected number of occurrences in each measurement period is $N/T$. Consequently, the probability $P_{nc}$ that a cell in a given STBF to which a given item $e$ maps is not collided, *i.e.*, no other item maps to this cell is given by the following equation.

$$P_{nc} = \left(1 - \frac{1}{m}\right)^{k(\frac{N}{T}-1)} \approx \left(1 - \frac{1}{m}\right)^{k\frac{N}{T}} \tag{3}$$

If the given item $e$ appears in $t$ out of the $T$ measurement periods, then the expected number of cells in the group of item $e$ during the identification phase will be $t \times P_{nc}$ (note: for now, we have ignored the mingling due to hash-print collisions; we will consider that shortly in Section 4.1.2). Consequently, the number of encoded bits available in these cells for PIE to use to recover the ID of item $e$ is equal to $r \times t \times P_{nc}$ Using the expression for the decoding failure probability of Raptor codes from Equation (1), the hash-mapping collision survival probability $P_m$ of PIE to recover IDs with length $l$ is calculated by the following equation.

$$P_m = 1 - P_{df}(r \times t \times P_{nc}; l) \tag{4}$$

Note that to calculate the value of $P_m$, we need the value $N$ of the total number of occurrences of all items in Equation (3), which is not directly known. Next, we present our maximum likelihood based estimation scheme to calculate the estimate $\tilde{N}$ of $N$.

Let $Z_{i0}$, $Z_{i1}$, and $Z_{iC}$ represent the number of empty, singleton, and collided cells, respectively, in the STBF of measurement period $i$ containing $m$ cells. The likelihood that the number of distinct items seen during the measurement period $i$ and stored in the corresponding STBF is $n_i$ is given by the following likelihood function.

$$L(n_i) = \left((1 - \frac{1}{m})^{kn_i}\right)^{Z_{i0}} \times \left(kn_i\frac{1}{m}(1 - \frac{1}{m})^{kn_i-1}\right)^{Z_{i1}} \times \left(1 - (1 - \frac{1}{m})^{kn_i} - kn_i\frac{1}{m}(1 - \frac{1}{m})^{kn_i-1}\right)^{Z_{iC}}$$

To simplify the equation above, we define $q = (1-\frac{1}{m})^{kn_i}$ and substitute it into this equation. We also substitute $Z_{iC} = m - Z_{i0} - Z_{i1}$ into this equation.

$$L(q) = q^{Z_{i0}} \times \left(\frac{q\ln\{q\}}{(m-1)(\ln\{1-\frac{1}{m}\})}\right)^{Z_{i1}} \times \left(1 - q - \frac{q\ln\{q\}}{(m-1)(\ln\{1-\frac{1}{m}\})}\right)^{m-Z_{i0}-Z_{i1}}$$

Taking the first-order derivative of the equation above and equating to 0, we get the sufficient condition that maximizes $L(q)$ as

$$Z_{i1} - qZ_{i1} + (Z_{i0} + Z_{i1})\ln\{q\} + q\{cZ_{i0} - (c+1)m\}\ln\{q\} - cmq\ln\{q^2\} = 0$$

where $c = \frac{1}{m-1} \times \frac{1}{\ln(1-1/m)}$. We numerically calculate the value $\tilde{q}$ of $q$ that satisfies the equation above, and is, thus, the maximum likelihood estimate of $q$. As $q = (1 - \frac{1}{m})^{kn_i}$, the maximum likelihood estimate $\tilde{n}_i$ of the number of distinct items seen during the measurement period $i$ is given by the following equation.

$$\tilde{n}_i = \frac{\ln\{\tilde{q}\}}{k\ln\{1-\frac{1}{m}\}} \tag{5}$$

As $\tilde{n}_i$ is the estimate of the number of distinct items seen during the measurement period $i$, the estimate $\tilde{N}$ of the total number of occurrences $N$ during $T$ measurement periods is given by $\tilde{N} = \sum_{i=1}^{T}\tilde{n}_i$.

### 4.1.2 Hash-print Collision Survival Probability

Calculating the hash-print collision survival probability $P_p$ is very challenging because we need to calculate the occurrence probability for a given number of cells in a mingled group. For this, we need to enumerate all possible combinations that lead to the same number of cells in a mingled group. For examples, if we have 90 cells in a mingled group, these 90 cells may have resulted from two items with 45 occurrences each, or from three items with 30 occurrences each, *etc.* Thus, this becomes a mathematically and computationally intractable combinatorics problem. To solve the problem of intractability, we present a novel bipartite approximation scheme to calculate $P_p$. For this, let $\lambda$ represent a threshold such that if an item $e^{\geq\lambda}$ with occurrences larger than or equal to $\lambda$ in the $T$ measurement periods makes a group mingled, then no item ID, except that for $e^{\geq\lambda}$, can be recovered from that mingled group. Next, we first calculate hash-print collision survival probability when at least one of the items in a mingled group has $\geq \lambda$ occurrences. We represent this probability with $P_p^{\geq\lambda}$. After this, we calculate hash-print collision survival probability when all of the items in a mingled group have $< \lambda$ occurrences. We represent this probability with $P_p^{<\lambda}$.

To calculate $P_p^{\geq\lambda}$, we first calculate the probability that an item $e^{\geq\lambda}$ with occurrences $\geq \lambda$ makes the group of any given item $e$ mingled, and then calculate the expected value of the number of items with occurrences $\geq \lambda$. The item $e^{\geq\lambda}$ makes the group of the item $e$ mingled when the item $e^{\geq\lambda}$ maps to a specific cell out of the $m$ cells in a given STBF and has the same hash-print as the item $e$. Considering only a single mapping hash function, this happens with probability $1/m \times 1/2^p$, where $p$ is the number of bits in the hash-print field of each cell. As there are $k$ mapping hash functions, the probability that the item $e^{\geq\lambda}$ does not make the group of the item $e$ mingled is $(1-1/m\times 1/2^p)^k$. Let $w_t$ represent the percentage of items that appear in $t$ out of $T$ measurement periods. The expected number of occurrences of items, *i.e.*, the average number of measurement periods during which an item appears, is given by $\sum_{t=1}^{T}(w_t \cdot t)$. Thus, the expected number of distinct items is $N/\sum_{t=1}^{T}(w_t \cdot t)$. Consequently, the expected number of items with occurrences $\geq \lambda$ is $N \times \sum_{t=\lambda}^{T} w_t / \sum_{t=1}^{T}(w_t \cdot t)$. As each item is mapped to $k$ cells, the probability that none of the items with occurrences $\geq \lambda$ make the group of the given item $e$ mingled is given by the following equation.

$$P_p^{\geq\lambda} = \left(1 - \frac{1}{m} \times \frac{1}{2^p}\right)^{kN\frac{\sum_{t=\lambda}^{T} w_t}{\sum_{t=1}^{T}(w_t \cdot t)}} \quad (6)$$

To calculate $P_p^{<\lambda}$, we treat each item with $t < \lambda$ occurrences as $t$ distinct items with a single occurrence each. With this notion, the expected number of items with occurrences $< \lambda$ is $N \times \sum_{t=1}^{\lambda-1}(w_t \cdot t)/\sum_{t=1}^{T}(w_t \cdot t)$ . Let $X_m$ be a random variable representing the number of such items mapped to a given cell. With $k$ mapping hash functions, it is straightforward to see that $X_m$ follows the binomial distribution $X_m \sim \text{Binom}(k \times N \times \sum_{t=1}^{\lambda-1}(w_t \cdot t)/\sum_{t=1}^{T}(w_t \cdot t), 1/m \times 1/2^p)$. PIE fails to recover the ID of an item $e$ from the mingled group only when the number of all cells from other items $e^{<}$ with occurrences $< \lambda$ in that group is less than the threshold $g_T$. Thus, $P_p^{<\lambda}$ is given by the following equation.

$$P_p^{<\lambda} = \sum_{j=0}^{g_T} \binom{kN\frac{\sum_{t=1}^{\lambda-1}(w_t \cdot t)}{\sum_{t=1}^{T}(w_t \cdot t)}}{j} \times \left(\frac{1}{m} \times \frac{1}{2^p}\right)^j \times \left(1 - \frac{1}{m} \times \frac{1}{2^p}\right)^{kN\frac{\sum_{t=1}^{\lambda-1}(w_t \cdot t)}{\sum_{t=1}^{T}(w_t \cdot t)} - j} \quad (7)$$

The overall hash-print collision survival probability $P_p$ is simply the product of $P_p^{\geq\lambda}$ and $P_p^{<\lambda}$, *i.e.*,

$$P_p = P_p^{\geq\lambda} \times P_p^{<\lambda}. \quad (8)$$

**Discussion:** Setting an appropriate value for $\lambda$ is a critical issue to calculate $P_p$ using our bipartite approximation scheme. An appropriate way to set $\lambda$ is to put it equal to $g_T/P_{nc}$ because the expected number of cells in a mingled group by an item with occurrences $\geq \lambda$ is $\lambda \times P_{nc}$, and this number must not be greater than the threshold $g_T$ for PIE to be able to successfully decode the IDs of other items in the mingled group. The value of $\lambda$ can also be selected after performing an empirical study with different values of $\lambda$ in the range [2 $g_T$]. The rationale behind developing a bipartite approximation scheme lies in the skewed distribution of item occurrence as well as the small value for $g_T$. The distributions of item occurrences in practical applications typically follows Zipf or "Zipf-like" distribution, which are highly skewed [23]. We observed this trend in the traces used for evaluation of PIE as well as shown in Figure 3(a). We observe from this figure that almost 75% of all occurrences are contributed by items with individual occurrences no more than 4. Similarly, about 40% of all occurrences are by item with individual occurrence of just 1. As typically $\lambda$ is proportional to $g_T$ (*e.g.*, $\lambda = g_T/P_{nc}$), and $g_T$ is a small number, $\lambda$ is also small. Consequently, such a value of $\lambda$ does not introduce much error when we treat an item with $t < \lambda$ occurrences as $t$ items with a single occurrence each. Thus, the overall error in the calculation of $P_p$ due to the bipartite approximation is very small.

#### 4.1.3 False Negative Rate Estimation

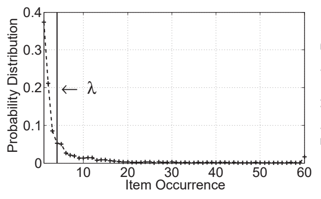
An item with $t$ occurrences, where $T_{th} \leq t \leq T$, can be recovered if and only if it survives both hash-mapping collisions as well as hash-print collisions, *i.e.*, the probability of the successful recovery of the ID of an item is $P_m \times P_p$, where $P_m$ and $P_p$ are calculated in Equations (4) and (8), respectively. The expected probability of successful recovery $P_{sr}$ for an item at a specific cell can be derived by taking into account all possible long-duration items as shown in the following equation.

$$P_{sr} = \frac{\sum_{t=T_{th}}^{T} w_t \times P_m}{\sum_{t=T_{th}}^{T} w_t} \times P_p \quad (9)$$

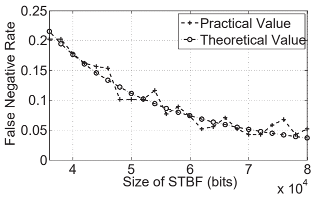
As PIE fails to recover an item only when it cannot be recovered from any of its $k$ mapped cells, the FNR is given by the following equations.

$$P_{FN} = (1 - P_{sr})^k \quad (10)$$

To see the accuracy of our proposed calculation of FNR, we conducted a simulation study and compared the FNR obtained through this study with the FNR calculated theoretically using the Equation (10). Figure 3(b) plots the FNR calculated from the simulation study as well as the FNR calculated theoretically using Equation (10). For this simulation study, we used the following parameters: $N$=59752, $T$=60, $T_{th}$=40, $l$=64, $k$=3, $g_T$=2, $r$=3 and $p$=1. We observe from this figure that the empirically calculated values of FNR lie very close to the theoretically calculated values of FNR. This supports our intuition from the discussion in Section 4.1.2 that the overall error in the calculation of $P_p$ due to the bipartite approximation is very small.



(a) Item occurrence prob. dist. for DC trace [4]



(b) FNR with bipartite approximation

**Figure 3: Dist. of item occurrences and the comparison of theoretical and empirical FNR**

### 4.2 False Positive Rate

Recall that PIE verifies each ID it recovers by first comparing the hash-print of the recovered ID with the hash-prints stored in the cells of the group and then comparing the hash-print with the hash prints in the cells at locations $h_1(ID), \cdots, h_k(ID)$. False positive rate (FPR) refers to the ratio of non-persistent items, which pass the two verification tests, to the number of all non-persistent items. There are two scenarios that give rise to false positives: (1) the recovered ID does not actually belong to any of the items seen at the observation point during the $T$ measurement periods; and (2) the recovered ID is exactly the same as some other persistent item. Next, we discuss these two scenarios.

**Scenario 1:** The item ID must be recovered from encoded bits that come from at least two different items from two different cells. During the verification, PIE will first examine whether the hash-print of the recovered ID is consistent with the one stored in the cells that corresponds to at least two different items at the considered position. The probability of error here equals $1/2^p$. During the verification, PIE will also check the other $k-1$ cells for recorded Raptor codes and hash-prints. This verification step will fail if the other $k-1$ cells are all collided, where the probability of a cell not being collided was calculated in Equation (3) and is represented by $P_{nc}$. Thus, the probability of failure of this second verification step is $(1/2^{r+p})^{2(k-1)P_{nc}}$. The false positive rate for this scenario is given by the following equation.

$$P_{FP1} = \left(\frac{1}{2^p}\right)\left(\frac{1}{2^{r+p}}\right)^{2(k-1)P_{nc}} = 2^{-\{p+2(r+p)(k-1)P_{nc}\}}$$

**Scenario 2:** Suppose the recovered item ID corresponds to an item with occurrence $t$ ($1 \leq t < T_{th}$), then there must be extra cells from other items that make the group mingled. Consider the worst case when there is only one other item making the group mingled. By similar analysis as for the first case, we can derive that the false positive rate for the second scenario is given by the following equation.

$$P_{FP2} = 2^{-\{r+(r+p)(k-1)P_{nc}\}} \tag{11}$$

Combining the analysis for these two scenarios, the FPR for the worst case is given by the following equation.

$$P_{FP} = \max\{P_{FP1}, P_{FP2}\}. \tag{12}$$

## 5. PIE – PARAMETER OPTIMIZATION

In this section, we calculate the optimal values of three parameters, the number of bits $r$ in the Raptor-code field of each cell, the number of bits $p$ in the hash-print field of each cell, and the number of cells $m$ in each STBF. The optimal values of $r$, $p$, and $m$ will minimize the FNR of PIE. Next, we will first formulate the optimization problem. After that, we will discuss how to solve this optimization problem to obtain the optimal values of $r$, $p$, and $m$.

### 5.1 Optimization Problem Formulation

Let $M$ represent the size of the entire STBF in bits. The value of $M$ can be provided by the system manager, who wants to allocate no more than $M$ bits of SRAM for recording information about the IDs of items. As the size of each cell in bits is $r+p+1$ and there are $m$ cells, the sum of the sizes of all these cells should at most be equal to $M$. Thus, the straightforward formulation of the optimization problem is as follows.

**Formulation 1:** Minimize $P_{FN}$ such that $m(r+p+1) = M$ and $r, m \in \mathbb{Z}^+, p \in \mathbb{Z}_0^+$.

From Equation (10), we observe that minimizing $P_{FN}$ is equivalent to maximizing $P_{sr}$ as long as the value of $k$, *i.e.*, the number of hash functions used to map an item to cells, is fixed. The expression for $P_{sr}$ in Equation (9) can be expanded and written as below.

$$P_{sr} = \left( \frac{\sum_{t=T_{th}}^{T} w_t \times (1 - P_{df}(r \times t \times P_{nc}; l))}{\sum_{i=T_{th}}^{T} w_i} \right) \left( P_p^{\geq\lambda} \times P_p^{<\lambda} \right)$$

The first term on the right hand side of the equation above depends on $P_{df}(r \times t \times P_{nc}; l)$. When $r \times t \times P_{nc} < l$, the value of $P_{df}(r \times t \times P_{nc}; l)$ is equal to 1. Thus, this first term equals 0. When $r \times t \times P_{nc} > l$, the value of $P_{df}(r \times t \times P_{nc}; l)$ exponentially approaches 0 and this first term almost achieves its maximum possible value of 1. Thus, to maximize $P_{sr}$, we should ensure that $r \times T_{th} \times P_{nc} \geq \kappa l$, where $\kappa$ is a preset constant such as $\kappa = 1.01$ or $\kappa = 1.1$. This serves as a new constraint, and our optimization problem can be reformulated as:

**Formulation 2:** Maximize $(P_p^{\geq\lambda} \times P_p^{<\lambda})$ such that $m(r+p+1) = M$; $r \times T_{th} \times P_{nc} \geq \kappa l$; and $r, m \in \mathbb{Z}^+, p \in \mathbb{Z}_0^+$.

Recall that $P_p^{\geq\lambda}$ and $P_p^{<\lambda}$ are calculated using Equations (6) and (7), respectively. Due to the complexity of these two equations, it is quite challenging to maximize their product. To address this challenge, we take a new approach to find a simpler expression for describing the hash-print collision survival probability $P_p$ that is feasible for parameter optimization. Let's consider the average number of cells corresponding to different IDs but with the same hash-print at a specific cell line. The probability that an item maps to a non-collided cell in this cell line and has the same hash-print as stored in some other cell in this cell-line is $1/2^p \times 1/m \times P_{nc}$. As the total number of occurrences of all items in the $T$ STBFs is $N$, the average number of mingling cells is $N \times 1/2^p \times 1/m \times P_{nc}$ Intuitively, the probability that the number of mingling cells in a group is less than the threshold $g_T$, *i.e.*, the hash-print collision survival probability $P_p^{\geq\lambda}$ and $P_p^{<\lambda}$, should be negatively correlated to the average number of mingling cells, which means maximizing the former is equivalent to minimizing the latter. As each item is mapped to $k$ positions, our optimization problem can be reformulated as follows.

**Formulation 3:** Minimize $(N \times 1/2^p \times 1/m \times P_{nc})^k$ such that $m(r+p+1) = M$; $r \times T_{th} \times P_{nc} \geq \kappa l$; and $r, m \in \mathbb{Z}^+, p \in \mathbb{Z}_0^+$.

Note that in this formulation, we do not see the term $w_t$, *i.e.*, the probabilistic distribution of occurrences of items. This is a highly desirable property of this formulation because now in implementing our proposed scheme in real applications we no longer need to know such probabilistic distribution a priori.

### 5.2 Calculating Optimal Values

LEMMA 1. *Given a fixed value of $m$ and $m(r+p+1) = M$ the false negative rate is minimized when the inequality $r \times T_{th} \times P_{nc} \geq \kappa l$ takes an equal sign.*

LEMMA 2. *Given a fixed value $r$ of the size of Raptor code field in cells and $m(r+p+1) = M$, the false negative rate is minimized when the inequality $r \times T_{th} \times P_{nc} \geq \kappa l$ takes an equal sign.*

LEMMA 3. *Given a fixed value of $p$ of the hash-print field in cells and $m(r+p+1) = M$, the false negative rate is minimized when the inequality $r \times T_{th} \times P_{nc} \geq \kappa l$ takes the equal sign.*

Combining Lemmas 1, 2 and 3, we get the following following theorem.

THEOREM 1. *The false negative rate is minimized if the inequality $r \times T_{th} \times P_{nc} \geq \kappa l$ takes the equal sign when $m(r+p+1) = M$, and the number of cells $m$ is equal to $\lfloor m^* \rfloor$, where $m^*$ satisfies the following equation*

$$\left(1 - \left(1 + \frac{kN}{T}\right)\frac{1}{m^*}\right) - \ln 2 \cdot \frac{1}{m^*}\left(1 - \frac{1}{m^*}\right) \times \left(M - \frac{\kappa l}{T_{th}}\frac{kN}{T}\left(1 - \frac{1}{m^*}\right)^{-\frac{kN}{T}-1}\right) = 0 \tag{13}$$

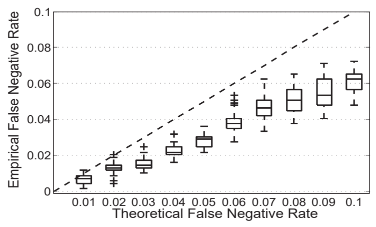
*and parameters $r$ and $p$ are calculated as below*

$$r = \left\lceil \frac{\kappa l}{T_{th}} \left(1 - \frac{1}{m}\right)^{-\frac{kN}{T}} \right\rceil \tag{14}$$

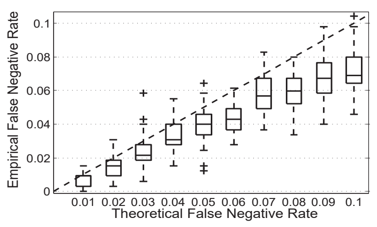
$$p = \left\lfloor \frac{M}{m} \right\rfloor - \left\lceil \frac{\kappa l}{T_{th}} \left(1 - \frac{1}{m}\right)^{-kN/T} \right\rceil - 1 \tag{15}$$

PROOF. We omit the proofs of Lemmas 1, 2, 3, and Theorem 1 to save space. □

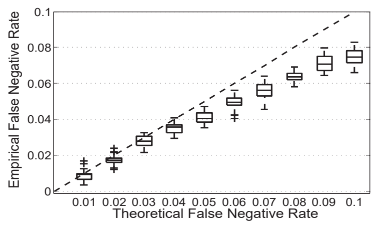
**Discussion** The parameter optimization scheme proposed in Theorem 1 has its inherent limitation. There exists a lower threshold for the objective function in Formulation 3, which



(a) CHIC

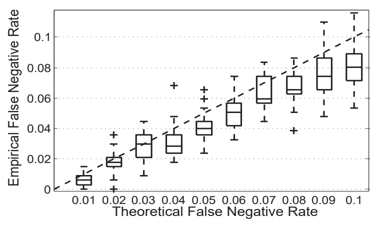


(b) DC

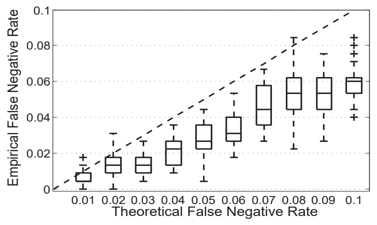


(c) ICSI

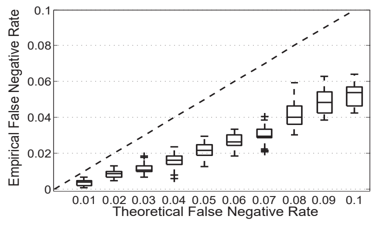
**Figure 4: Empirical false negative rate vs. theoretical false negative rate when $T_{th} = 40$**



(a) CHIC



(b) DC



(c) ICSI

**Figure 5: Empirical false negative rate vs. theoretical false negative rate when $T_{th} = 50$**

can be written in the form of $(\overline{\kappa} \times g_T)^k$ where $\overline{\kappa}$ is a constant that can be empirically estimated. When the value of the objective function in the Formulation 3 falls below $\overline{\kappa} g_T$, the hash-print collision survival probability is very close to 1, Therefore, further minimization at this point is not just meaningless, but could also hurt the overall optimality in some cases. To avoid this problem, we can simply enumerate all possible values of the length of hash-print $p$ from 0 to $l$ to compute $m$ and $r$ using the equations $m(r+p+1) = M$ and $r \times T_{th} \times P_{nc} = \kappa l$ and then substitute them into Equation (10) to calculate the false negative ratio. Finally, we pick the value of $p$ that leads to the minimum false negative ratio and use the corresponding values of $m$ and $r$. Overall, such process will terminate within $O(l)$ time, which is fairly fast as $l$ is typically anywhere from 16 bits to 64 bits.

## 6. PERFORMANCE EVALUATION

We implemented and extensively evaluated PIE along with two other schemes namely Invertible Bloom Filter (IBF) [9,14] and Count-Min (CM) sketch [8] in Matlab. Next, we first describe the three network traces that we used for evaluation. After that, we present our results from the evaluation of PIE's FNR and FPR and compare them with IBF and CM sketch.

### 6.1 Item Traces and Setup

**Item Traces:** To evaluate the performance of PIE, we use three real network traces CHIC [1], ICSI [21] and DC [4]. CHIC is a backbone header trace, published by CAIDA and collected in 2015, which includes the arrival times of packets at a 10GigE link interface along with the flow IDs associated with those packets. For this evaluation, we used HTTP flows from 6 minutes of packet capture. We treat each packet as an item and the 5-tuple flow ID of each packet as the item ID. ICSI is an enterprise network traffic trace collected at a medium-sized enterprise network. From this trace, we use TCP traces generated from 22 different ports in one hour of packet capture. DC is a data center traffic trace collected at a university data center collected for a little more than an hour. From this trace, we use TCP traces generated from one hour of packet capture. Table 1 reports the total duration, number of packets and number of distinct flows of these three network traces. Figure 6 shows the item occurrence probability distributions for three network traces.

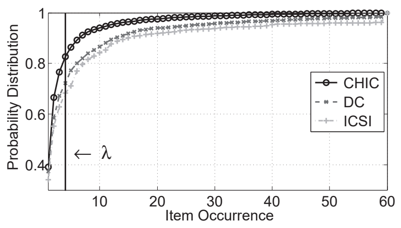


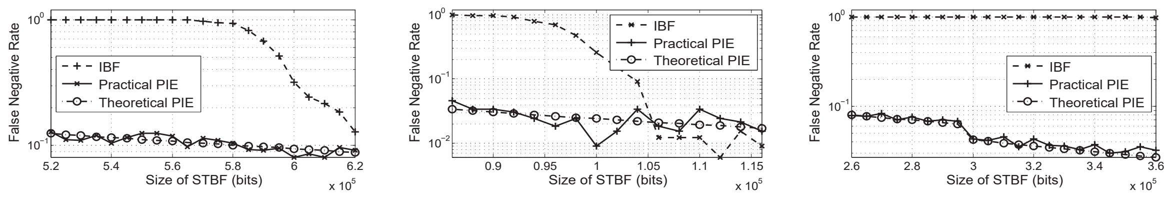
**Figure 6: Item occurrence prob. dist. for three traces**

**Table 1: Summary of Item Traces**

| Trace | Duration | # pkts | # flows |
|---|---|---|---|
| CHIC | 6 mins | 25.3M | 101374 |
| ICSI | 1h | 1.49M | 8797 |
| DC | 1h | 8.09M | 10289 |

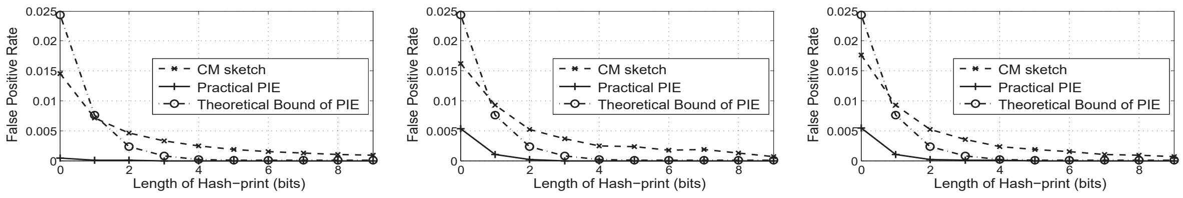
**System Parameters:** In our experiments, for each trace, we set the number of measurement periods as $T = 60$, the constant for Raptor code decoding $\kappa = 1.05$, the number of mapping hash functions $k = 3$, the mingling threshold $g_T = 1$, and the amount of memory allocated to build STBF $M = 600$kbits for CHIC, $M = 100$kbits for ICSI, and $M = 300$kbits for DC, respectively.

**Baseline Setup:** As IBF and CM sketch can not be directly applied to identify persistent items, we adapt them to make performance comparison possible. For IBF, instead of employing a single IBF throughout all measurement periods, we allocate one IBF to each measurement period. As IBF can not tell whether an incoming item is recorded or not, we need an additional conventional Bloom filter (BF) to store existence information of items. When an arriving item is not bound in the associated conventional BF, we add its information to both conventional BF and IBF. For CM sketch, we again need a conventional BF to avoid counting of multiple repetitions of the same item in any given measurement period. Furthermore, for fair comparison, we allocate the same amount of memory to build IBF or CM sketch as that to build PIE. Moreover, to the advantage of IBF and CM sketch, we neglect the storage cost of these assisting conventional BFs for both IBF and CM sketch. We



(a) CHIC (b) DC (c) ICSI

**Figure 7: False negative rate when $T_{th} = 40$**



(a) CHIC (b) DC (c) ICSI

**Figure 8: False positive rate when $T_{th} = 40$**

also ignore the space cost of CM sketch for recording IDs of persistent items, which would be considerable when the number of distinct items is large.

## 6.2 False Negative Rate

We first present the FNR of PIE. For this, we set a maximum desired value of FNR, calculate the optimal values of $r$, $p$, and $m$ for the maximum desired FNR as described in Section 5.2, perform the simulations, and compare the FNR obtained from the simulations with the maximum desired FNR. We repeat this process for multiple values of maximum desired FNR. After this, we compare the FNR of PIE with the FNR of IBF. We do not compare the FNR of PIE with the FNR of CM sketch because the FNR for CM sketch is always 0.

**FNR of PIE:** *Our results show that the average FNR of PIE calculated from simulations is always less than the maximum desired FNR.* We vary the maximum desired FNR from 0.01 to 0.1 and for each value of FNR, calculate the parameters $r$, $p$, and $m$ for the three traces. For each set of parameters, we randomly select part of the data with fixed size from each of three networks traces, and repeat the experiments 50 times. This gives us 50 values of FNR from simulations for each value of maximum desired FNR. We do all these experiments with two values for the threshold $T_{th}$, *i.e.*, $T_{th} = 40$ and $T_{th} = 50$; and plot the results in Figures 4 and 5, respectively. We plot the 50 values of FNR obtained from simulations for each value of maximum desired FNR as a box-plot. On each box-plot, the central mark is the median, the edges of the box are the 25-th and 75-th percentiles, and the crosses are the outliers. The dashed lines in each figure corresponds to the function $y = x$. From these figures, we observe that the average values of FNR obtained from simulations are always less than the maximum desired FNR. This indicates that PIE can be configured to achieve any desired value of FNR by appropriately adjusting the parameters $r$, $p$, and $m$.

**Comparison with IBF:** *Our results show that the average FNR of PIE is almost twice an order of magnitude smaller than the FNR of IBF.* Figure 7 plots the FNRs of PIE and IBF for $T_{th} = 40$. We do not plot figures for $T_{th} = 50$ due to space constraints. These figures also plot the FNR of PIE calculated theoretically using Equation (10). We observe that the theoretically calculated FNR closely matches the empirically calculated FNR of PIE. We also observe from these figures that FNRs of both PIE and IBF decrease with the increase in the number of bits $M$ allocated to their corresponding data structures. From our experiments, we calculated that on average, PIE achieves 19.5 times smaller FNR compared to IBF. We observe from these figures that the FNR of IBF remains almost 1 when $M$ is small, and then drops rapidly as the value of $M$ increases. The reason behind this observation is that the recovering process of IBF resembles the "peeling process" of finding a 2-core in random hypergraphs, and its success probability of recovering rockets when the percentage of cells that store only one item, which is determined by $M$, is sufficiently large. Note that IBF achieves a better performance compared to PIE only when the size of its data structure is significantly large. For example, IBF needs $1.12 \times 10^5$ bits of storage space for DC trace, which is so large that it equals half the space required to store all occurrences of items along with their original IDs. This cost may not be acceptable for real applications.

## 6.3 False Positive Rate

For FPR, we compare the FPR of PIE with the FPR of CM sketch. We do not compare with IBF because the FPR for IBF is always 0. For the simulations to obtain results for this section, we fix the length of Raptor code field in cells of STBF to 2, and set the size of each counter in CM sketch to 8 bits, which is sufficient to record items that appear in all 60 measurement periods.

*Our results show that FPR of PIE is at least* 426.1 *times less than the FPR of CM sketch.* Figure 8 plots the FPR of PIE and CM sketch along with the theoretical bound on FPR calculated using Equation 12. We only plot figures for $T_{th} = 40$ and not for $T_{th} = 50$ due to space constraints. We observe that the empirically obtained FPR of PIE is always less than the theoretically calculated bound. We also observe from these figures that the FPR of PIE becomes 0 when the length of hash-print field exceeds 5 for $T_{th} = 40$ for all three traces. Even in the worst case, the FPR of PIE is just 24.9% the FPR of CM sketch. These figures further show that the FPRs of both PIE and CM sketch improve with increasing $p$. This happens because larger $p$ means fewer hash-print collisions and more success during verification steps for PIE and more counters for CM sketch.

We also observe from our experiments that overall, the FPR of CM sketch increase with $T_{th}$ while the FPR of PIE decrease with $T_{th}$. As described in [8], the estimate occurrence $\hat{i}$ of an item obtained through CM sketch is subject to $\hat{i} \le i + B \times N$ with some probability $\alpha$. Though the absolute error of estimation, *i.e.*, $B \times N$, is roughly stable, the number of eligible items decreases with an increasing $T_{th}$. This gives rise to an increasing FPR for CM-sketch. On the contrary, for PIE, with a large $T_{th}$, there are potentially more cells recording a given item, which provides more opportunities for verification of item ID and thus reduces the FPR.

## 7. CONCLUSION

The key contribution of this paper is in proposing a scheme to identify persistent items. The key technical novelty of this paper is in proposing the space-time Bloom filter, which uses Raptor codes and hash-prints to efficiently store information about the IDs of items in such a way that the IDs of persistent items can be recovered with the desired FNR. The key technical depth of this paper is in calculating the FNR and FPR and using them to obtain optimal values of system parameters. Our theoretical analysis and experimental results show that PIE always achieves the desired FNR and its FNR is always less than the FNRs of prior schemes adapted for persistent item identification. PIE is scalable in that it is designed to store information about the IDs of all items in a single data structure, the STBF, in a given measurement period and periodically transfers the contents of the STBF to permanent storage.

## Acknowledgment

Alex X. Liu is also affiliated with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA. This work is partially supported by the National Science Foundation under Grant Numbers CNS-1318563, CNS-1524698, and CNS-1421407, and the National Natural Science Foundation of China under Grant Numbers 61502229, 61472184, and 61321491, and the Jiangsu High-level Innovation and Entrepreneurship (Shuangchuang) Program.

## 8. REFERENCES

[1] The caida ucsd anonymized 2011 internet traces. `http://www.caida.org/data/overview.`

[2] C. C. Aggarwal. An introduction to sensor data analytics. In *Managing and Mining Sensor Data*, pages 1–8. Springer, 2013.

[3] M. H. S. Bangalore. Resource adaptive technique for frequent itemset mining in transactional data streams. *IJCSNS*, 12(10):87, 2012.

[4] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proc. IMC*, pages 267–280. ACM, 2010.

[5] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.

[6] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming*, pages 693–703. Springer, 2002.

[7] G. Cormode and M. Hadjieleftheriou. Finding the frequent items in streams of data. *Communications of the ACM*, 52(10):97–105, 2009.

[8] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[9] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese. What's the difference?: efficient set reconciliation without prior context. In *Proc. SIGCOMM*, volume 41, pages 218–229. ACM, 2011.

[10] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proc. CoNEXT*, pages 75–88. ACM, 2014.

[11] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Data stream mining. In *Data Mining and Knowledge Discovery Handbook*, pages 759–787. Springer, 2010.

[12] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Quicksand: Quick summary and analysis of network data. Technical report.

[13] F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, and D. Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *Recent Advances in Intrusion Detection*, pages 326–345. Springer, 2009.

[14] M. T. Goodrich and M. Mitzenmacher. Invertible bloom lookup tables. In *Proc. Allerton*, pages 792–799. IEEE, 2011.

[15] N. Immorlica, K. Jain, M. Mahdian, and K. Talwar. Click fraud resistant methods for learning click-through rates. In *Internet and Network Economics*, pages 34–45. Springer, 2005.

[16] Y. Li, H. Wu, T. Pan, H. Dai, J. Lu, and B. Liu. Case: Cache-assisted stretchable estimator for high speed per-flow measurement. In *Proc. INFOCOM.*

[17] H. Liu, Y. Lin, and J. Han. Methods for mining frequent items in data streams: an overview. *Knowledge and information systems*, 26(1):1–30, 2011.

[18] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. VLDB*, pages 346–357. VLDB Endowment, 2002.

[19] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Database Theory-ICDT 2005*, pages 398–412. Springer, 2005.

[20] S. Moro, R. Laureano, and P. Cortez. Using data mining for bank direct marketing: An application of the crisp-dm methodology. In *Proc. ESM*, pages 117–121. Eurosis, 2011.

[21] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A first look at modern enterprise traffic. In *Proc. IMC*, pages 15–28, 2005.

[22] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[23] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Transactions on Networking*, 12(2):219–232, 2004.

[24] A. Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52(6):2551–2567, 2006.

[25] R. M. Transport. Raptorq forward error correction scheme for object delivery. Technical report, IETF Internet Draft, 2011.

[26] Q. Xiao, Y. Qiao, M. Zhen, and S. Chen. Estimating the persistent spreads in high-speed networks. In *Proc. ICNP*, pages 131–142. IEEE, 2014.