2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)

Threshold-Based Widespread Event Detection

You Zhou[†‡] Yian Zhou[†‡] Shigang Chen[†]
[†]Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL, USA
[‡]Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA, USA
Email: youzhou@cise.ufl.edu yianzhou@google.com sgchen@cise.ufl.edu

***Abstract*—Widespread event detection is a fundamental network function that has many important applications in cybersecurity, traffic engineering, and distributed data mining. This paper introduces a new probabilistic threshold-based event detection problem, which is to find all events that appear in any w-out-of-a monitors with probabilistic guarantee on false positives, where a is the total number of monitors and the threshold $w(\leq a)$ is a positive integer parameter that can be arbitrarily set, according to specific application requirements. We develop an efficient threshold filter solution and its improved versions, which combine Bloom filters, counting Bloom filter, threshold filter and compressed filters in a series of encoding and filtering steps, providing tradeoff between detection accuracy and communication overhead. We theoretically optimize the system parameters in the proposed solutions to minimize the communication overhead under the constraint of probabilistic detection guarantee. Extensive simulations demonstrate the practical viability of the proposed solutions in their ability of finding widespread events in a large network with few false positives and low communication overhead.**

I. Introduction

Detection of widespread events across a network is a fundamental function that has many important applications in cybersecurity, traffic engineering, and trend monitoring [1], [2], [3], [4], [5]. During an outbreak of Internet worms [6], knowing which subnets have detected infection will enable coordinated defense and monitoring of how infection is spreading. During a large-scale DDoS attack such as the 2016 Mirai attack [7], knowing which subnets are out of Internet service and which are not would help diagnosis quickly narrow down toward the critical failing points that brought down service access. With a distributed deployment of honeypots that catch intrusion attempts [1], knowing multiple attempts at different honeypots sets distributed attacks from targeted ones. Generally speaking, widespread attacking events should raise alerts at higher levels for immediate attention and prioritized responses. Significance of widespread event detection goes beyond security applications. In a network that experiences frequent congestions, knowing whether this is a single point of congestion or a widespread problem at multiple routers provides guideline on how to expand link capacities and restructure network topology in order to relieve congestion. A global Internet search company may also benefit from detecting which phrases were widely searched at its servers distributed at different geographical locations. Profiling widespread interest over time helps the company learn various trends among the Internet users in terms of news development, commercial product popularity, political opinions, etc.

There are two system models for widespread event detection: the coordinator model [4] and the peer-to-peer model [5]. The coordinator system model consists of a central coordinator and a set of distributed monitors, and the P2P model does not have a coordinator. The monitors may be intrusion detection systems (IDS), firewalls, honeypots, routers, or servers, each producing logs about events that have happened locally. For example, some of the logs produced from the intrusion detection systems (i.e., monitors) may be designed for distributed worm detection as follows: At the end of every reporting period, each IDS sends the coordinator all ⟨source address, destination port⟩ pairs that it sees in the packet stream. Here, each address/port pair is an event. If many IDSes observe the same pair, this "widespread" event signals a possible distributed scanning. If this event persists over time with more and more other similar events (i.e., scanners) join in, it signals a possible ongoing worm propagation [6]. In another example, servers of an Internet search company may want to compare their search records to find popular search phrases. Here, servers are monitors and search phrases are events.

In the coordinator model [4], monitors report their data to the coordinator, which process the received data and communicate the results back to the monitors in order to help them identify widespread events. It takes one round trip delay. In the P2P model [5], the monitors exchange data amongst themselves to find widespread events. It takes numerous round trips, each round having some monitor pairs to exchange their data. Technically, all prior solutions [4], [5] are designed to only detect *the widespread events that are observed by all monitors*, which seriously limits their practical use. In the previous example of distributed scanning detection, we do not have to observe a source/port pair at all monitors. If it appears in an unusually large number of monitors, an anomaly alert could be issued. Similarly, we do not have to see a search phrase appears in all search servers in order to declare that it is popular.

This paper introduces a generalized problem called threshold-based event detection, which is to find all events that appear in any w out of a monitors, where a is the total number of monitors and the threshold $w(\leq a)$ is a positive integer parameter that can be arbitrarily set, according to specific application requirements. The prior art [4] is a special case of the generalized problem with $w = a$, and their simplified solutions cannot be used to solve the general problem, which requires us to resort different techniques to deal with false positives and control communication overhead.

We focus on the coordinator system model in this paper

2575-8411/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDCS.2019.00047

to avoid long delay from the P2P model where monitors exchange data in many sequential rounds. The coordinator model takes only one round of communication between all monitors and the coordinator. However, as all monitors report their data to the same coordinator, it creates a communication bottleneck [4]. Consider the previous example where every IDS sends the coordinator its source/port pairs, which can number in millions, and this is just one type of events that IDS will report. With a large number of IDSes, the combined traffic volume to the coordinator can be overwhelming. In the Internet search example, as all servers send their sets of search records to the coordinator, the combined traffic can again be overwhelming. Transmitting all raw event data to the coordinator may cause tremendous network traffic, strain its access link, and thus degrade the performance of widespread event detection.

To reduce communication overhead, we relax the problem from exact widespread event detection to probabilistic detection, such that we do not have to send raw data but instead minimize communication through lossy encoding and compression. Our probabilistic detection requires that *all events that appear in any w-out-of-a monitors must be detected with zero false negative*, while there may be a small number of false positives, meaning that an event appearing in fewer than w monitors may also be reported, with a probability smaller than ε, which is a system parameter that can be set arbitrarily small, allowing tradeoff between detection accuracy and communication cost. We believe widespread event detection with probabilistic guarantee is acceptable to most applications. In the previous examples, as long as we can successfully detect all distributed scanners and find all real popular search phrases, a small number of false positives are nuisance but will not nullify the usefulness of the system.

The contributions of this paper are summarized as follows.

• We introduce a generalized threshold-based formulation for widespread event detection with probabilistic guarantee. To the best of our knowledge, this is the first work on the generalized widespread event detection problem, with the prior art being a special case (with $w = a$).

• We develop an efficient threshold filter solution and its improved variants, which perform distributed computation between monitors and a coordinator, combining Bloom filters, counting Bloom filter, threshold filters and compressed filters in a series of encoding and filtering steps, which together reduce communication overhead between the monitors and the coordinator significantly, while ensuring probabilistically guaranteed event detection.

• We mathematically reveal the tradeoff between detection accuracy and communication overhead, prove that the probabilistic guarantee is achieved, and theoretically optimizes the system parameters in the proposed threshold filter solutions to minimize the communication overhead.

• We perform extensive simulations to demonstrate that our solutions achieve probabilistic guarantee of threshold-based widespread detection with high efficiency and low communication cost.

II. System Model and Problem Statement

A. System Model

We consider a distributed network monitoring system consisting of a number a of distributed monitors and a central coordinator. These monitors are deployed at selected vantage points in the network. They independently monitor and record events, which can be customized depending on applications as suggested in the introduction. The monitors report information of their event sets to the coordinator at a pre-defined frequency or upon query. The coordinator is responsible for collecting, combining and synthesizing data from different monitors to form a global view. It then informs the monitors of the combined information for further actions. This general model has been adopted in many prior work on distributed monitoring [8], [9], [10], [11].

B. Problem Definition

Let $X = \{x_1, x_2, \ldots, x_a\}$ be the set of monitors, and E_i the set of events observed by monitor x_i, $1 \le i \le a$. Let E_* be the union of all event sets, i.e., $E_* = \bigcup_{i=1}^{a} E_i$. Consider an arbitrary event $e \in E_*$. The number of monitors that observe the event is called the *frequency* of the event, denoted as $f(e)$. An event e is called a w-widespread event if it is observed by w or more monitors, i.e., $f(e) \ge w$, where w is a threshold value set by the user based on specific application needs, and it is distributed by the coordinator to all monitors.

The *threshold-based widespread event detection* problem is for each monitor x_i, $1 \le i \le a$, to find all w-widespread events in its event set E_i so that it may react accordingly based on pre-set policies, e.g., blocking the source addresses in distributed scanning, logging the traffic of widespread activities for further analysis, or placing the results of widespread web searches in cache for quicker response. Let W_i be the set of w-widespread events in E_i,

$$W_i = \{e : e \in E_i, f(e) \ge w\}. \quad (1)$$

The sets of w-widespread events at different monitors may not be identical because any w-widespread event may be observed by some monitors but not others. We denote the union of W_i, $1 \le i \le a$, as W_*.

Exactly finding W_i for each monitor can be very expensive when there are many monitors and each monitor has numerous events. If the function of widespread event detection is performed frequently for real-time reaction and all monitors need to send their events (or event identifiers) to the coordinator for precise identification, the communication overhead can be extraordinarily high, particularly for the coordinator. Now if we relax the requirement by performing detection approximately, we may be able to reduce the overhead to a small fraction of the brute-force approach of sending the raw event sets to the coordinator. As have been argued earlier, many applications can tolerate inaccurate identification of w-widespread events. For example, if there are false positives where some non-widespread events are classified as widespread, the applications of logging widespread activities or caching widespread web searches should work fine as long as the false positive ratio is made sufficiently small.

Even for applications that require precise identification of w-widespread events, an approximation solution may still help as it can be used as a pre-processing step to filter out most non-widespread events and thus allow the monitors to only transmit the much-reduced approximate sets of widespread events to the coordinator.

Let $\widehat{W}_i$ be the approximate w-widespread event set identified at monitor x_i. This paper considers the *probabilistic threshold-based widespread detection*, which aims to find $\widehat{W}_i$ at each monitor x_i with probabilistic guarantee that consists of the following two requirements:

- *Completeness requirement:* $\forall i \in [1, a], W_i \subseteq \widehat{W}_i$.

We require every member in W_i must belong to the approximate w-widespread event set $\widehat{W}_i$. That is, for an arbitrary monitor x_i, it must find all w-widespread events.

- *False positive ratio requirement:* $\forall i \in [1, a]$, $\forall e \in E_i - W_i$, $\varepsilon_i = Prob\{e \in \widehat{W}_i\}$ is the false positive ratio at monitor x_i. Let $\varepsilon = \max_{1 \le i \le a} \{\varepsilon_i\}$ be the worst-case false positive ratio. We require that $\varepsilon \le \varepsilon^*$, where $\varepsilon^* \in (0, 1)$ is a false positive upper bound pre-defined based on application needs.

For an event at monitor x_i that is not w-widespread, the false positive ratio ε_i is the probability that this event is misclassified to $\widehat{W}_i$, which should be upper-bounded by ε^*. For example, when ε^* is set to 0.01, we require that the false positive ratios of all monitors are less than 0.01. Clearly, a smaller value of ε ($\le \varepsilon^*$) means more accurate detection results.

Since all monitors communicate with the coordinator, the communication amount of traffic received/sent by the coordinator is a primary performance concern. Therefore, we strive to reduce the communication of the coordinator. Meanwhile, we should also reduce the communication and computation overhead at each monitor.

We formulate two optimization goals in our solutions. For the first one, given an upper bound of communication overhead that the system can tolerate, we minimize the worst-case false positive ratio at any monitor, i.e., $\varepsilon = \max_{1 \le i \le a} \{\varepsilon_i\}$. For the second one, given a preset accuracy requirement ε^*, we minimize the total amount of communication overhead.

C. Naive Solution

As we mentioned earlier, a naive solution to calculate W_i for each monitor x_i is using raw data transmission. Each monitor stores the observed events locally and transmit those events to the coordinator at the end of each reporting period. The coordinator combines all received data, identifies the w-widespread events for each monitor, and then notifies each monitor its w-widespread event set. Suppose b bits are required to represent each event. Then the total amount of data that the coordinator receives from all monitors is $\sum_{i=1}^{a} b|E_i|$, and the total amount of data that the coordinator sends to all monitors is $\sum_{i=1}^{a} b|W_i|$. Therefore, the total communication overhead for the coordinator is $\sum_{i=1}^{a} b(|E_i|+|W_i|)$. When the number of events in each monitor or the number of monitors is large, this solution incurs significant network traffic at the coordinator, which is not applicable in practice.

III. Related Work

The emergence of large-scale cooperative monitoring systems has drawn much research attention over the recent years. Some researches focus on applying specific aggregate functions for the data collected from distributed monitors, including threshold function [8], [9], discovering top-k [12], heavy hitters and quantiles [13], [14], and set expression cardinality estimation [15], [16]. Other research focuses on monitoring distributed data streams [10], [11]. Also related is distributed database join. Mackert et al. proposed Bloomjoin [17] to reduce communication cost for joining two distributed data sets. Michael et al. [18] studied the intersection of multiple lists in a distributed system without a central coordinator, where a series of two-list join is performed sequentially among distributed lists. This sequential approach takes long time when there are a lot of lists. Chen et al. [5] performs sequential joins among event monitors arranged in a hypercube structure. Most related is the work by Cai et al. [4] on widespread event detection. However, their techniques are limited to detect widespread events that are observed at all monitors. To the best of our knowledge, we are the first to introduce and solve the probabilistic threshold-based widespread event detection problem, where widespread events are defined based on a parameter w that can be set based on application needs.

IV. Threshold Filter Solution

We present a threshold filter solution (TFS) as the starting point for solving the probabilistic threshold-based widespread event detection.

A. Bloom Filter and Counting Bloom Filter

A Bloom filter [19] is a bit array used to encode memberships of a set. In our context, to encode an event in the set from a monitor, we pseudo-randomly select k bits in the filter through k hash functions, and set these bits to ones. For membership lookup of a given event, we map it to k bits in the filter using the same hash functions. If all k bits are ones, we classify that this event belongs to the set; otherwise it is not in the set.

A Bloom filter does not have false negative, meaning that it never mis-classifies a member event in the set as a non-member. But it may have false positive, meaning that it may mis-classify a non-member as a member. The probability for this to happen is called false positive ratio, which should be made sufficiently small. Let n be the number of events in the set, and m be the number of bits in the filter. The false positive ratio P_f is

$$P_f = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{\frac{-kn}{m}}\right)^k. \qquad (2)$$

It is well known that when $k = \ln 2 \cdot \frac{m}{n}$, P_f is minimized to $(\frac{1}{2})^k = (0.6185)^{\frac{m}{n}}$. For example, when $m = 8n$, the false positive ratio is minimized to 0.02 when k takes the optimal value of 5. Under the optimal k value, each bit in the filter has a 50% probability to be 0.

A counting Bloom filter [20], [21] replaces each bit in Bloom filter with a counter, which is initialized to zero. When encoding an event, we hash the event to k counters and increase those counters by one.

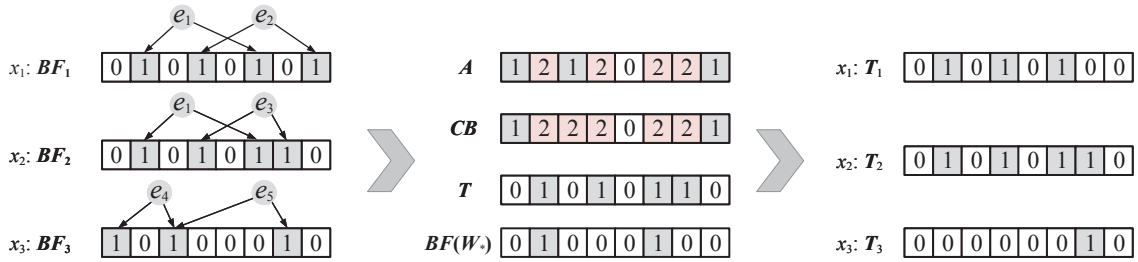


Fig. 1: An example of the threshold filter solution.

B. Main Idea

Bloom filters provide compact representation of event sets. Sending them instead of raw data from monitors to the coordinator helps reduce communication overhead. Counting Bloom filters use counters to keep event multiplicity information, which is what the widespread event detection needs. One idea is for each monitor x_i to encode its event set E_i in a counting-Bloom filter and for all monitors to send their counting-Bloom filters to the coordinator, which combines the filters into a single counting-Bloom filter for E_* by adding the corresponding counters in all received filters. However, this approach is not very communication-efficient because the size of a counting-Bloom filter is multiple times that of a Bloom filter, depending on the counter size which has to accommodate the largest counter value in the whole filter.

To ensure communication efficiency, our approach will **not** use a separate counting Bloom filter at each monitor. Instead we let each monitor x_i use a simple Bloom filter BF_i to encode E_i and send the filter to the coordinator. After receiving such Bloom filters from all monitors, the coordinator combines them into a single counter filter A by adding the corresponding bits in the received filters. A will be different from the counting-Bloom filter (CB) for E_i, which we will show through an example. The coordinator then turns A back into a bit filter called the threshold filter T, which is then filtered by BF_i before being sent to monitor x_i for detecting all w-widespread events in E_i.

C. Design Details

At the end of each reporting period or upon query from the coordinator, every monitor x_i encodes its event set E_i in a Bloom filter BF_i of m bits, and then sends BF_i to the coordinator. Figure 1 gives an example with $k = 2$, $a = 3$, $w = 2$, and $W_* = \{e_1\}$. The value of m controls the communication overhead. We will discuss how to set its value in the next section where we analyze the system parameters.

When the coordinator receives all a Bloom filters $\{BF_i, 1 \le i \le a\}$, it combines them into a single counter array A by adding them up: $A[j] = \sum_{i=1}^{a} BF_i[j]$, $1 \le j \le m$, where $A[j]$ refers to the jth counter in A and $BF_i[j]$ refers to the jth bit in the filter BF_i. A is different from the counting-Bloom filter (CB) for the union E_* of $\{BF_i, 1 \le i \le a\}$, which is also shown in the middle of Figure 1 under A.

Next the coordinator converts the counter array A into a bit array called *master threshold filter* T as follows: If $A[j] \ge w$, $T[j] = 1$; otherwise, $T[j] = 0$, where $T[j]$ refers to the jth bit of T, $1 \le j \le m$. The intuition is that a w-widespread event will set the same k bits at the Bloom filters from w or more monitors. When we add the filters to form A, those corresponding counters will be at least w. When we convert those counters back to ones (to save space and communication overhead), the resulting bit array would become a filter that encodes the set W_* of all w-widespread events. Unfortunately, T is not exactly a Bloom filter for W_* because a bit in T may be set to one by "noise" in A, instead of by events in W_*, which will be explained at the end of this section. That is, T may have more ones than the Bloom filter for W_*, which is shown as $BF(W_*)$ below T in the example of Figure 1.

Finally, the coordinator produces an *individual threshold filter* T_i for each monitor x_i by performing bitwise AND on T and BF_i. It then sends T_i to monitor x_i. The idea is that we shall remove ones in T where the corresponding bits in BF_i are zeros, meaning that no event in E_i is mapped to those bits. Compare T and T_1 in Figure 1. The 7th bit in T is one, but after bitwise AND with BF_1, the bit becomes zero as no event in $E_1 = \{e1, e2\}$ is mapped to this bit. Upon receiving T_i, monitor x_i performs membership lookup for all its events in E_i. An event is classified as a w-widespread event if its k bits in T_i (which the event is hashed to) are all ones; we denote such classified events as a subset $\widehat{W}_i \in E_i$. Note that the false positive ratio of filter T_i is denoted as ε_i in Section II-B.

For example, in Figure 1, when x_1 looks up for e_1 and e_2 in filter T_1, it will find that only e_1 is a member and thus $\widehat{W}_1 = \{e_1\}$, which is correct. Similarly, x_2 finds $\widehat{W}_2 = \{e_1, e_3\}$, including a false positive e_3, and x_3 finds that $\widehat{W}_3 = \{\}$, which is also correct.

Following the structure of TFS, it is straightforward to show by the following proposition that T_i does not cause false negative, i.e., $W_i \subseteq \widehat{W}_i$, which means the completeness requirement is met.

Proposition 1: $\forall e \in W_i, e \in \widehat{W}_i$ after execution of TFS, for $1 \le i \le a$.

Proof: For an arbitrary event e in W_i, it must be observed in w' monitors, where $w \le w' \le a$. Denote these w' monitors as $x_{i_1}, x_{i_2}, \ldots, x_{i_{w'}}$. The k bits that e maps to will all be ones in Bloom filters $BF_{i_1}, BF_{i_2}, \ldots, BF_{i_{w'}}$. Thereby the corresponding k counters in the counter array A must all be

at least w'. Since $w' \geq w$ and $e \in E_i$, those k counters will be converted to ones in the corresponding k bits in the master threshold filter T, as well as in its threshold filter T_i. When the monitor x_i performs the membership lookup for the event e, it will find all k bits for e are set to ones in T_i, and thus include it in $\widehat{W}_i$. ■

The false positive ratio requirement will be met by setting the system parameters m and k appropriately in the next section, where m controls the communication overhead. Setting these Bloom-filter parameters is tricky as we show below that the intuitive thought of using optimal Bloom filters BF_i, $1 \leq i \leq a$, as described in Section IV-A, may result in poor performance.

We use simulations to test the performance of TFS under optimal Bloom filters BF_i, $1 \leq i \leq a$. The number a of monitors is set to 10. The number n of events in each monitor is set to 10,000. We set the length m of each Bloom filter BF_i to $8n$ or $15n$. We use the optimal k value for each filter, i.e., $k = \ln 2 \cdot \frac{m}{n}$. The event frequency $f(e)$, $e \in E_*$, follows a zipf distribution in $[1, a]$. The simulation is repeated for 100 times to obtain statistical results.

We never observe any false negative in the simulations, which confirms Proposition 1 empirically. Figure 2 presents the performance of TFS, where x-axis shows the threshold value w from 2 to 10, and the y-axis shows the worst-case false positive ratio ε, which is the largest among ε_i, $1 \leq i \leq a$, i.e., the false positive ratio of individual threshold filter T_i. The figure shows that ε becomes larger when w becomes smaller. For example, with $m = 8n$, as w decreases from 10 to 2, ε increases from near zero to 0.86. Clearly, optimizing BF_i does not necessarily make T_i perform well, particularly for a small w, which means we need to set the parameters of BF_i differently.

We provide an intuitive explanation of the above observation. For optimal BF_i with $k = \ln 2 \cdot \frac{m}{n}$, the probability for any of its bits to be one (or zero) is 0.5. When we add BF_i, $1 \leq i \leq a$, to produce A, any counter in A has an average value of $\frac{a}{2}$, which represents a pervasive noise level in A. Now consider the k counters for an event that is not w-widespread, if the threshold w is small, the noise in those counters (at a level of $\frac{a}{2}$) can easily go beyond the threshold, causing a false positive, since the corresponding k bits in T_i are all ones (with those k counters greater than or equal to w). Intuitively, we will need to reduce the value of k for BF_i and thus the probability for bits in BF_i to be ones, in order to reduce the noise level in A and thus the number of ones in T, which in turn helps reduce the false positive ratios of individual threshold filters T_i.

V. Optimal Threshold Filter Solution

In this section, we analyze the optimal parameter setting, and the resulting optimal threshold filter solution is OTFS.

A. Parameter Optimization

There are two types of optimization. The first is to choose the optimal number k of hash functions that minimize the worst-case false positive ratio of the threshold filters T_i,

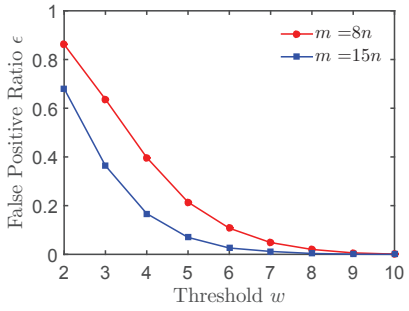


Fig. 2: False positive ratios of TFS under optimal BF_i.

subject to a given size m of BF_i, $1 \leq i \leq a$, which is set to limit the communication overhead between the monitors and the coordinator. This optimization problem is fundamentally different from setting the optimal k for a Bloom filter to minimize the *same* filter's false positive ratio. We are setting the k value for BF_i to minimize the worst-case false positive ratio of T_i, $1 \leq i \leq a$. The second optimization is to choose the optimal number k of hash functions that minimizes the size m of BF_i, i.e., the communication overhead between the monitors and the coordinator, subject to an upper bound ε^* for the false positive ratios of T_i, $1 \leq i \leq a$.

The second optimization is to choose the optimal number k that minimizes the size m of BF_i, i.e, the communication overhead between the monitors and the coordinator, subject to an upper bound ε^* for the false positive ratios of T_i, $1 \leq i \leq a$.

1) First Optimization OTFS-I: Let $n^* = \max_{1 \leq i \leq a} \{|E_i|\}$, and p_i denote the probability that a bit in BF_i is 0, $1 \leq i \leq a$. According to [22], we have

$$p_i = \big(1 - \frac{1}{m}\big)^{k|E_i|} \approx e^{-\frac{k|E_i|}{m}}. \tag{3}$$

Let p^* be the value of p_i when $|E_i| = n^*$. That is,

$$p^* = \big(1 - \frac{1}{m}\big)^{kn^*} \approx e^{-\frac{kn^*}{m}}. \tag{4}$$

Clearly, p^* is the minimum value of p_i,

$$p_i \geq p^*, \ \ \forall 1 \leq i \leq a. \tag{5}$$

Consider an arbitrary event e whose frequency $f(e)$ is smaller than w. We analyze the worst-case probability ε for it to be mis-classified as a w-widespread event. Event e is hashed to the same k bits in T_i and BF_i, $1 \leq i \leq a$. False positive happens when these k bits in T_i are all ones, i.e., the corresponding k counters in A are greater than or equal to w. Consider an arbitrary one of these k bits, and let j be the bit index. Let P be the probability of $A[j] \geq w$. It is easy to see that when P is maximized,

$$\varepsilon = P^k. \tag{6}$$

$A[j]$ is the sum of $BF_i[j]$, $1 \leq i \leq a$. Because e appears at $f(e)$ monitors, the corresponding Bloom filters have ones at index j, which means $A[j]$ is at least $f(e)$. Let Π be the set of Bloom filters from monitors where e does not appear. Hence, P is the probability for at least $w - f(e)$ filters in Π to have

one at index j, each happening with a probability $1-p_i$. P is maximized under the following conditions: i) $f(e) = w-1$, i.e., it suffices for any filter in Π to have one at index j; ii) $p_i = p^*$, i.e., the probability for any filter in Π to have one at index j takes the maximum value $1-p^*$ from (5). That is, the worst case happens when all event sets have the same size n^*. Under these conditions, the probability for all $(a-w+1)$ filters in Π to have zero at index j is $(p^*)^{a-w+1}$. Hence, in the worst case,

$$P = 1 - (p^*)^{a-w+1}. \tag{7}$$

From (6) and (7), we have

$$\varepsilon = \left(1 - (p^*)^{\alpha}\right)^k, \tag{8}$$

where we define *joint differential* $\alpha = a - w + 1$.

Upon query from the coordinator, all monitors report their event sizes $|E_i|$, $1 \le i \le a$, to the coordinator, which finds the maximum event set size n^*. If the coordinator has a communication constraint that limits the amount of the traffic between itself and the monitors, it sets the value of m such that the total traffic, $m \cdot a$ bits, does not exceed the constraint. We can assume that $m > n^*$, which is reasonable for any Bloom filter of practical use because otherwise the false positive ratio would be too high (62% or more). The coordinator finds the optimal value k that minimizes the false positive ratio ε of T_i in (8) as follows: Since $p^* = e^{-\frac{kn^*}{m}}$ by definition, we have $k = \frac{-m\ln(p^*)}{n^*}$. Apply it to (8), we have ε as a function of p^*,

$$\varepsilon = \left(1 - (p^*)^{\alpha}\right)^{\frac{-m\ln(p^*)}{n^*}} = \left(e^{-\ln(p^*)\cdot\ln(1-(p^*)^{\alpha})}\right)^{\frac{m}{n^*}}. \tag{9}$$

Let $\beta = -\ln(p^*)\cdot\ln(1-(p^*)^{\alpha})$. Since $m > n^*$, in order to minimize ε in (9), we shall minimize β. Taking derivative of β with respect to p^*, we have

$$\frac{d\beta}{dp^*} = \frac{(p^*)^{\alpha}\ln\left((p^*)^{\alpha}\right) - \left(1-(p^*)^{\alpha}\right)\ln\left(1-(p^*)^{\alpha}\right)}{p^*\cdot(1-(p^*)^{\alpha})}. \tag{10}$$

It is easy to check that $\frac{d\beta}{dp^*}$ is 0 when $(p^*)^{\alpha} = \frac{1}{2}$. Furthermore, from the symmetry of the expression of $\frac{d\beta}{dp^*}$, it is easy to check that $\frac{d\beta}{dp^*}$ is negative for $0 < (p^*)^{\alpha} < \frac{1}{2}$ and positive for $\frac{1}{2} < (p^*)^{\alpha} < 1$. Hence, β (also ε) is minimized when $(p^*)^{\alpha} = \frac{1}{2}$. Because $k = \frac{-m\ln(p^*)}{n^*}$, we have the optimal value for k as

$$k = \ln 2 \cdot \frac{m}{\alpha n^*}. \tag{11}$$

Applying it to (8), we have the optimal false positive ratio ε,

$$\varepsilon = (\frac{1}{2})^{\ln 2\cdot\frac{m}{\alpha n^*}}. \tag{12}$$

In practice, we must use at least of one hash function. Hence,

$$k = \max\{1, \lfloor \ln 2 \cdot \frac{m}{\alpha n^*} \rfloor\}. \tag{13}$$

The optimal threshold filter solution that chooses k by (13), subject to the value of m, is named as OTFS-I, which attempts to minimize the worst-case false positive ratio of T_i, $1 \le i \le a$. After finding the value k, the coordinator sends m and k to all monitors, which will construct BF_i, $1 \le i \le a$.

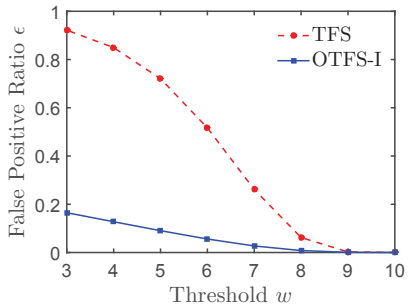


Fig. 3: False positive ratios of TFS and OTFS with $m = 30n^*$.

2) Second Optimization OTFS-II: Suppose the coordinator sets an upper bound ε^* for the false positive ratio of T_i, $1 \le i \le a$. Under this requirement, we want to find the optimal value k that minimizes the size m of BF_i, i.e., the communication overhead between the monitors and the coordinator. The optimal threshold filter solution that achieves the above goal is OTFS-II, whose parameters are set as follows. From earlier analysis, we know that under any value of m, the best possible false positive ratio is given by (12) under the optimal k in (11). Replacing ε in (12) with its upper bound ε^*, we derive the minimum value of m for such a false positive ratio,

$$m = -\frac{\alpha\ln\varepsilon^*}{(\ln 2)^2}\cdot n^*. \tag{14}$$

Applying (14) to (11), we have

$$k = -\ln\varepsilon^*/\ln 2. \tag{15}$$

Because k must be a positive integer, we set

$$k = \max\{1, \lfloor -\ln\varepsilon^*/\ln 2 \rfloor\}. \tag{16}$$

The coordinator sends the computed m and k to all monitors.

B. Numerical Results

We use simulations to compare TFS using optimal BF_i (Section IV) and OTFS-I. We set the value m to be $15n^*$. Figure 3 presents the average false positive ratios of individual threshold filters T_i with respect to the threshold w. Clearly, OTFS-I outperforms TFS with much smaller false positive ratios. When the threshold w decreases, the false positive ratio moves up, which is also expected from (12). When w is very small (such as 3), the false positive ratio is 0.18 when $m = 30n^*$. To reduce the ratio, we will have to further raise m (thus the communication overhead). Can we increase m without causing higher communication overhead? This is what the compressed threshold filter solution will do next.

VI. Compressed Threshold Filter Solution

A. Motivation

Consider OTFS-I. Its number of hash functions, computed from (11) as $k = \ln 2 \cdot \frac{m}{\alpha n^*}$, is smaller than that of an optimal Bloom filter, i.e., $k = \ln 2 \cdot \frac{m}{n^*}$ for encoding n^* events. For an optimal Bloom filter, each bit has an equal chance to be one or zero. The filter cannot be further compressed. The communication overhead for its transmission is m bits.

However, BF_i in OTFS-I is constructed with a smaller value of k. Hence, BF_i is not an optimal Bloom filter. In fact, each of its bits has a greater chance to be zero, which means the filter can be compressed to reduce communication overhead [23].

Suppose we have a constraint for the communication overhead to be bounded by $2az$, where z is per-filter overhead. If we send the filters BF_i uncompressed, then $m = z$, which is the case in the previous section. But if we compress the filters before sending, we disconnect m from the overhead, and can therefore increase it to reduce the false positive ratio; see Figure 3. Similar to OTFS, there are also two types of optimization. The first is to choose the optimal values of k and m that minimize the worst-case false positive ratio of the individual threshold filters T_i, subject to a given maximum size z of BF_i after compression, $1 \leq i \leq a$. The second optimization is to choose the optimal values of k and m that minimizes the maximum size z of BF_i after compression, subject to an upper bound ε for the false positive ratios of T_i, $1 \leq i \leq a$. The size m of BF_i before compression can be much larger than z.

We point out that the idea of compressed Bloom filters was proposed in [23], but our work is completely different. We optimize the performance of threshold filters T_i when the Bloom filters BF_i are compressed before transmission. Threshold filter is a new concept in this paper, not in [23].

B. Design of CTFS

Upon query from the coordinator, all monitors report their event sizes $|E_i|$, $1 \leq i \leq a$, to the coordinator, which finds the maximum event set size n^*. Depending on which optimization will be used, the coordinator determines the values of the system parameters k and m (see the next subsection). It then sends k and m to all monitors. Each monitor x_i encodes its event set E_i in a Bloom filter BF_i of m bits, and compresses the filter before transmitting it to the coordinator. Upon receipt of all compressed filters, the coordinator decompresses them to recover BF_i, $1 \leq i \leq a$. It adds them to produce A and then the master threshold filter T as described in Section IV-C. It produces individual threshold filters T_i by performing bitwise AND on T and BF_i. It compresses T_i before sending it to monitor x_i. T_i has no more ones than BF_i (which contains more zeros than ones); in fact, all bits of ones in T_i must be ones in BF_i. Therefore, the size of T_i will be no greater than that of BF_i after compression. After monitor x_i receives its compressed threshold filter, it decompresses it to recover T_i. Monitor x_i performs membership lookup for all its events in E_i. An event is classified as a w-widespread event if its k bits in T_i (which the event is hashed to) are all ones.

CTFS is similar to OTFS except for the compression / decompression component. This seemly small difference in operation has significant impact on performance. CTFS performs much better than OTFS under the same overhead constraint or the same false positive requirement. Technically, the addition of compression / decompression makes the optimization problems much harder, which we will address next.

C. Parameter Optimization

From the compression limit in Shannon's source coding theorem [24], for an m-bit Bloom filter BF_i in which each bit has a probability p_i to be 0, it can be compressed down to $mH(p_i)$ bits, where p_i is given by (3) and $H(p_i) = -p_i \log_2(p_i) - (1-p_i)\log_2(1-p_i)$ is the binary entropy function. In the worst case, $mH(p_i)$ is maximized to $mH(p^*)$ when p_i takes the minimum value p^* as specified in (4).

1) First Optimization CTFS-I: Given the maximum event-set size n^* and an upper bound z for the size of any compressed filter, we want to decide the values of k and m that minimize ε, subject to $mH(p^*) \leq z$, where $z > n^*$. We know that a larger filter size m helps reduce false positive. So we choose the largest value by letting

$$m = \frac{z}{H(p^*)}. \tag{17}$$

Combining it with (9) for the worst-case false positive ratio of threshold filters, we have

$$\begin{aligned} \varepsilon &= \left(1-(p^*)^\alpha\right)^{\frac{-z\ln p^*}{n^* H(p^*)}} = \left(e^{-\frac{\ln(p^*)\cdot\ln(1-(p^*)^\alpha)}{H(p^*)}}\right)^{\frac{z}{n^*}} \\ &= \left(\exp\left(\frac{\ln(p^*)\cdot\ln(1-(p^*)^\alpha)}{(\log_2 e)(p^*\ln(p^*)+(1-p^*)\ln(1-p^*))}\right)\right)^{\frac{z}{n^*}}. \end{aligned} \tag{18}$$

Define γ to be the exponent inside the parentheses,

$$\gamma = \frac{\ln(p^*)\cdot\ln(1-(p^*)^\alpha)}{p^*\ln(p^*)+(1-p^*)\ln(1-p^*)}. \tag{19}$$

Since $z > n^*$ and $\log_2(e) > 0$, in order to minimize ε, we shall minimize the exponent γ. Appendix A of the supplement material [25] shows that γ decreases as p^* increases when $\alpha > 1$. To minimize γ, we need to maximize p^*. Initially, let's assume that p^* can take any value from 0 to 1. The proof of the following lemma can be found in Appendix A of the supplement material [25].

Lemma 1: Given any value $\alpha \geq 1$, γ is minimized in the limiting case to -1 as p^* goes to 1.

Theorem 1: For an arbitrary value $\alpha \geq 1$, the minimal false positive ratio ε is $(0.5)^{\frac{z}{n^*}}$.

Proof: According to Lemma 1, γ is minimized in the limiting case as p^* goes to 1. The value of ε is minimized when γ is minimized. In this limiting case, γ goes to -1, and ε goes to $e^{-\frac{z}{n^*\log_2 e}} = (0.5)^{\frac{z}{n^*}}$. ■

In theory, we can achieve a false positive to $(0.5)^{\frac{z}{n^*}}$ by approaching p^* to 1. However, P^* cannot take arbitrary values. It is a function of integer k. In order for p^* to approach to 1, according to (4), k should approach to zero, which is not possible. In practice, if we set k to its smallest value of 1, p^* takes its maximum practically-possible value $e^{-\frac{n^*}{m}}$, which in turn minimizes γ (and thus ε), since γ monotonically decreases as p^* increases. Consequently, with the use of compressed filters, the optimal value of k should be set to one, for the optimization problem of minimizing the false positive ratio of threshold filters under the constraint of per-filter communication overhead of z bits. A small value for k has an additional benefit that construction of the Bloom filters BF_i and lookup of the threshold filters T_i require fewer hash operations, which saves computation overhead.

Although compression and decompression require additional computation, they are one-time cost, and there are extremely efficient algorithms to do so [26].

With $k = 1$, we can compute the optimal value of m, together with the corresponding value of p^*, from the equations of (17) and (4). Applying $k = 1$ to (4), we have

$$p^* = e^{-\frac{n^*}{m}}. \tag{20}$$

Applying (17) to (20) to eliminate m, we have

$$p^* = e^{-n^* H(p^*)/z}. \tag{21}$$

We can solve the above equation numerically for the value of p^*. We then apply this value back to (17) for the optimal value of m. We use CTFS-I to denote the compressed threshold filter solution that uses the optimal values of k and m as computed above. Applying the computed value of p^* to (18), we can find the worst-case false positive ratio of the threshold filters under the optimal values of k and m.

2) Second Optimization CTFS-II: Given the maximum event-set size n^* and an upper bound ε^* for the false positive ratios of the threshold filters, we want to decide the values of k and m that minimize the worst-case compressed filter size z for transmission. From (17) and (4), we have the worst-case filter size after compression as

$$z = -kn^* H(p^*)/\ln(p^*). \tag{22}$$

Replacing ε in (8) with ε^*, we have

$$k = \ln \varepsilon^* / \ln(1 - (p^*)^\alpha). \tag{23}$$

Applying (23) to (22), we have

$$\begin{aligned} z &= -\tfrac{H(p^*)\ln \varepsilon^* n^*}{\ln(p^*)\ln(1-(p^*)^\alpha)} \\ &= \frac{p^*\ln(p^*) + (1-p^*)\ln(1-p^*)}{\ln(p^*)\cdot\ln(1-(p^*)^\alpha)} n^* \log_2 e \ln \varepsilon^*. \end{aligned} \tag{24}$$

According to (19), the fraction term on the right is $\frac{1}{\gamma}$. Hence,

$$z = \frac{1}{\gamma} n^* \log_2 e \ln \varepsilon. \tag{25}$$

The value of γ is negative. We know earlier that it decreases as p^* increases. Hence, its absolute value increases as p^* increases. To minimize z, we want to maximize the absolute value of γ, which means maximizing the value of p^*. The values of p^* and k are connected through (8), for a given value of ε^*. According to (8), in order to maximize p^*, we shall minimize k to its smallest possible value of one. Applying $k = 1$ and (4) to (8), we simplify it to

$$\varepsilon^* = 1 - (p^*)^\alpha = 1 - e^{-\frac{\alpha n^*}{m}}. \tag{26}$$

$$m = -\alpha n^* / \ln(1 - \varepsilon^*). \tag{27}$$

Applying (27) and $k = 1$ to (4), we can find the value of p^* under the optimal values of m and k:

$$p^* = e^{\frac{\ln(1-\varepsilon)}{\alpha}}. \tag{28}$$

Finally, applying this p^* value and $k = 1$ to (22), we will get the minimized filter size z after compression in the worst case.

VII. Simulations

In this section, we evaluate the performance of OTFS and CTFS through simulations. Since this is the first work on threshold-based widespread event detection, there is no prior work to compare with. Therefore, we will use our baseline approach, TFS with optimal Bloom filters BF_i (Section IV), as a benchmark for comparison. The most related work [4] is a special case of our generalized solution. In fact, when $w = a$, the proposed solution reduces nicely to [4]. Namely, the two will have the same performance. However, the method in [4] cannot handle any case of $w < a$.

A. Simulation Settings

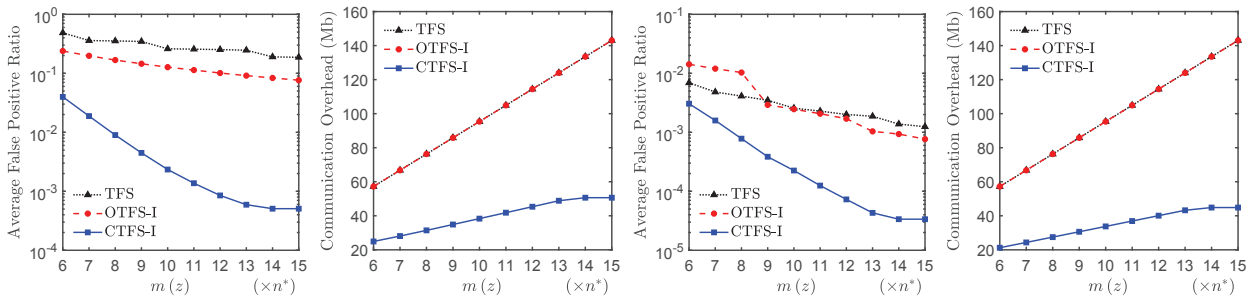
We consider two types of optimization as described in Section V and Section VI. The first optimization is to minimize the worst-case false positive ratio ε of the threshold filters when per-filter transmission overhead is bounded by m for OTFS and z for CTFS. The second optimization is to minimize the communication overhead concentrated at the coordinator, subject to a preset false-positive ratio upper bound ε^*.

The default number of monitors a is set to 10 though we will vary it for scalability study. The number $|E_i|$ of events at each monitor is randomly chosen from $[100000, 500000]$ with max value $n^* = 500000$. Each event has a 64-bit unique identifier. The frequency $f(e)$ of event $e \in E_*$ follows a zipf-like distribution in $[1, a]$. When studying the first optimization, we vary the per-filter overhead bound m (z), with m for TFS/OTFS and z for CTFS. We compare the average false positive ratio of all monitors under different solutions, including TFS with optimal Bloom filters, OTFS-I and CTFS-I, with respect to m (z). For the second optimization, we vary the false-positive ratio upper bound ε^*, and compare the overall communication overhead at the coordinator under different solutions, including OTFS-II and CTFS-II, with respect to ε^*. TFS with optimal Bloom filters is not included here because it cannot guarantee a false positive upper bound. Each simulation is repeated for 100 times to obtain statistical results.

B. Performance w.r.t. Per-filter Overhead Bound

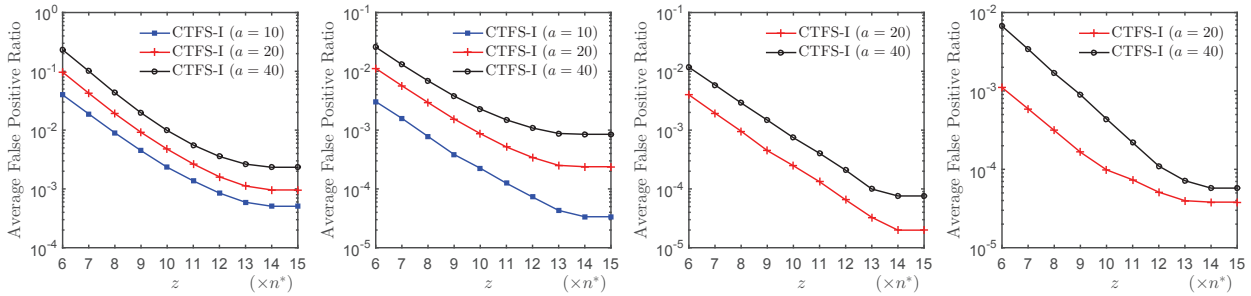
In the first set of simulations, we compare the performance of TFS, OTFS-I and CTFS-I with respect to the per-filter transmission overhead bound m (z), where m is for TFS/OTFS-I and z is for CTFS-I. The simulation results for threshold $w = 3$ and $w = 8$ are given in Figure 4, where x-axis shows the per-filter transmission overhead bound m (z) in units of n^*. Since similar comparisons are observed for other threshold values w, we omit those to save space.

Figure 4a and Figure 4c compare the average false positive ratios of TFS, OTFS-I and CTFS-I when $w = 3$ and $w = 8$, respectively. OTFS and CTFS are based on TFS, thereby they all meet the completeness requirement in Section II-B. Thus, a smaller false positive ratio means more accurate detection. When the threshold value is large (e.g., $w = 8$ as shown in Figure 4c), all solutions can achieve very accurate detection with false positive ratios consistently below 0.01. When the threshold value is relatively small (e.g., $w = 3$ as shown



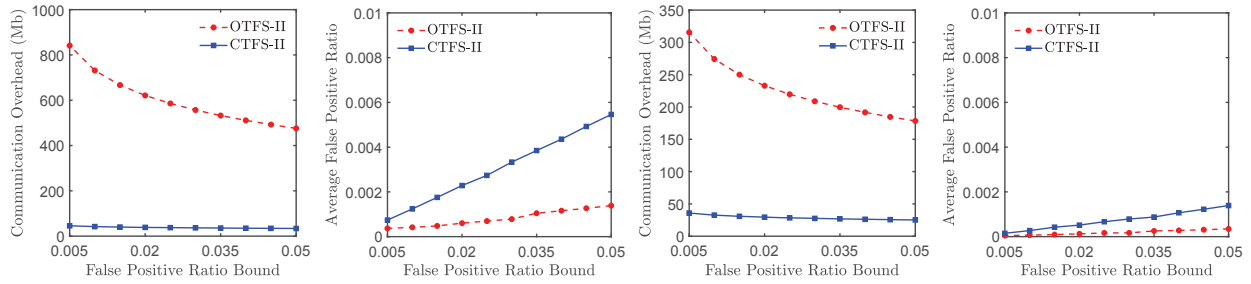
(a) $w = 3$ (b) $w = 3$ (c) $w = 8$ (d) $w = 8$

Fig. 4: Performance comparison of TFS, OTFS-I and CTFS-I with respect to per-filter communication overhead m (z) when $a = 10$



(a) $w = 3$ (b) $w = 8$ (c) $w = 13$ (d) $w = 18$

Fig. 5: False positive ratio of CTFS-I with respect to per-filter communication overhead bound z under different monitor number a



(a) $w = 3$ (b) $w = 3$ (c) $w = 8$ (d) $w = 8$

Fig. 6: Performance comparison of OTFS-II and CTFS-II with respect to false positive ratio bound ε^*

in Figure 4a), the false positive ratios of TFS and OTFS-I become large, while that of CTFS-I stays small, ranging from 0.0004 to 0.03, which is expected because compression makes CTFS-I more effective in information transfer for the same communication overhead. One can also see that the accuracy of our solutions all improve when m (z) increases, which is consistent with our earlier theoretical analyses.

Figure 4b and Figure 4d compare the actual communication overhead of different solutions. Clearly, CTFS-I incurs much less communication overhead than TFS and OTFS-I under the same per-filter bound. This is because CTFS-I compresses filters before transmission, and the compression ratios vary with different sparsity in the filters. More sparse filters will have higher compression ratios, thereby saving more communication overhead. These results demonstrate the superior performance of CTFS-I.

C. Detection Accuracy w.r.t. Number of Monitors

Next, we evaluate the performance of CTFS-I with respect to different number of monitors, a. We vary a from 10 to 20 to 40. The results are presented in Figure 5. From the figure, under the same threshold w, the detection accuracy (i.e., false positive ratio) becomes slightly worse as a increases. For example, in Figure 5b, when $w = 8$ and $z = 10n^*$, the false positive ratio is 0.0002 when $a = 10$, 0.0008 when $a = 20$, and 0.0028 when $a = 40$. Intuitively, a larger number of monitors will incur more noise in the filter A, which will cause more false positives, as explained in Section IV-C.

D. Performance w.r.t. False Positive Ratio Bound

In the last set of simulations, we compare the performance of OTFS-II and CTFS-II with respect to the false-positive ratio bound ε^*. The simulation results with $w = 3$ and

$w = 8$ are shown in Figure 6, where ε^* varies from 0.005 to 0.05. Figure 6b and Figure 6d present the average false positive ratios among all monitors. We can see that the average false positive ratios for OTFS-II and CTFS-II are smaller than the preset upper bound, which conforms to their design goals. Figure 6a and Figure 6c compare the communication overhead of the two solutions, where the overhead is defined as the amount of data that is received and sent by the coordinator. CTFS-II incurs far less communication overhead than OTFS-II under every threshold value w. For example, when $w = 3$ and $\varepsilon = 0.03$, the overhead of CTFS-II is only 38Mb, far less than 583Mb of OTFS-II, thanks to its communication reduction by compression. This confirms the superior performance of CTFS-II as its compression design intends to achieve.

VIII. Conclusion

In this paper, we introduce and formalize a new problem of probabilistic threshold-based widespread event detection in large networks. We first propose a threshold filter solution (TFS) based on highly compact Bloom filters, from which we develop two improved solutions, optimal threshold filter solution (OTFS) and compressed threshold filter solution (CTFS). OTFS theoretically optimizes the communication cost by selecting appropriate parameters for Bloom filters. CTFS further introduces compression and decompression for transmitting Bloom filters. We not only theoretically analyze their performance, but also perform extensive simulations to complement the theoretic analysis. The simulation results demonstrate that our solutions can efficiently provide accurate threshold-based widespread event search results with low communication cost.

IX. Acknowledgment

This work is supported in part by the National Science Foundation under grant CNS-1719222, STC-1562485, National Institute of Health under grant NIH R01GM118737, and a grant from Florida Center for Cybersecurity.

References

[1] L. Spitzner, "The Honeynet Project: Trapping the Hackers," *IEEE Security & Privacy*, vol. 99, no. 2, pp. 15–23, 2003.
[2] J. Ginsberg, M. H. Mohebbi, R. S. Patel, L. Brammer, M. S. Smolinski, and L. Brilliant, "Detecting Influenza Epidemics Using Search Engine Query Data," *Nature*, vol. 457, no. 7232, p. 1012, 2009.
[3] "Google trends." [Online]. Available: https://www.google.com/trends/
[4] Z. Cai, M. Chen, S. Chen, and Y. Qiao, "Searching for Widespread Events in Large Networked Systems by Cooperative Monitoring," *Proc. of IEEE ICNP*, pp. 123–133, 2015.
[5] J. Chen, Z. Cai, and S. Chen, "Efficient Distributed Joint Detection of Widespread Events in Large Networked Systems," pp. 1–6, 2016.
[6] P. K. Manna, S. Chen, and S. Ranka, "Inside the Permutation-Scanning Worms: Propagation Modeling and Analysis," pp. 858–870, 2010.
[7] "Mirai." [Online]. Available: https://en.wikipedia.org/wiki/Mirai_(malware)
[8] R. Keralapura, G. Cormode, and J. Ramamirtham, "Communication-Efficient Distributed Monitoring of Thresholded Counts," *Proc. of the 2006 ACM SIGMOD*, pp. 289–300, 2006.
[9] I. Sharfman, A. Schuster, and D. Keren, "A Geometric Approach to Monitoring Threshold Functions over Distributed Data Streams," *ACM Transactions on Database Systems (TODS)*, vol. 32, no. 4, p. 23, 2007.
[10] M. Garofalakis, D. Keren, and V. Samoladas, "Sketch-based Geometric Monitoring of Distributed Stream Queries," *Proc. of the VLDB Endowment*, vol. 6, no. 10, pp. 937–948, 2013.
[11] A. Friedman, I. Sharfman, D. Keren, and A. Schuster, "Privacy-Preserving Distributed Stream Monitoring," *Proc. of NDSS*, 2014.
[12] B. Babcock and C. Olston, "Distributed Top-K Monitoring," *Proc. of ACM SIGMOD*, pp. 28–39, 2003.
[13] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi, "Holistic Aggregates in a Networked World: Distributed Tracking of Approximate Quantiles," *Proc. of the ACM SIGMOD*, pp. 25–36, 2005.
[14] K. Yi and Q. Zhang, "Optimal Tracking of Distributed Heavy Hitters and Quantiles," *Algorithmica*, vol. 65, no. 1, pp. 206–223, 2013.
[15] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi, "Distributed Set-Expression Cardinality Estimation," *Proc. of VLDB*, pp. 312–323, 2004.
[16] P. Jesus, C. Baquero, and P. S. Almeida, "A Survey of Distributed Data Aggregation Algorithms," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 381–404, 2015.
[17] L. F. Mackert and G. M. Lohman, "R* Optimizer Validation and Performance Evaluation for Distributed Queries," *Proc. of VLDB*.
[18] L. Michael, W. Nejdl, O. Papapetrou, and W. Siberski, "Improving Distributed Join Efficiency with Extended Bloom Filter Operations," *Proc. of IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pp. 187–194, 2007.
[19] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[20] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, p. 281293, 2000.
[21] J. Aguilar-Saborit and P. Trancoso, "Dynamic Count Filters," *Proc. of ACM SIGMOD*, 2006.
[22] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet mathematics*, vol. 1, no. 4, pp. 485–509, 2004.
[23] M. Mitzenmacher, "Compressed Bloom Filters," *IEEE/ACM Transactions on Networking (TON)*, vol. 10, no. 5, pp. 604–612, 2002.
[24] "Shannon's source coding theorem." [Online]. Available: https://en.wikipedia.org/wiki/Shannon's_source_coding_theorem
[25] "Supplement Material. Link will be given in final version."
[26] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic Coding Revisited," *ACM Transactions on Information Systems (TOIS)*, vol. 16, no. 3, 1998.