2015 IEEE 23rd International Conference on Network Protocols

Searching for Widespread Events in Large Networked Systems by Cooperative Monitoring

Zhiping Cai[†] [‡] Min Chen[‡] Shigang Chen[‡] Yan Qiao[§]

[†] College of Computer, National University of Defense Technology, 410073 Changsha, Hunan, P.R.China

[‡] Department of Computer & Information Science & Engineering University of Florida, Gainesville, FL 32611, USA

[§] Google Inc., CA 94043, USA

***Abstract*—Searching for widespread events in large networks is a fundamental function that underlies many important applications of distributed anomaly detection, traffic measurement, online data mining, etc. This function can be performed by a cooperative monitoring system consisting of a central coordinator and a number of monitors that are deployed at a set of vantage points. We formulate a network primitive function, called *multi-monitor joint detection*, which is to find the common events observed by all or a given subset of monitors during each measurement period. It is a challenging problem because large-scale cooperative monitoring can generate tremendous communication overhead. Therefore, it is critical to design a solution for multi-monitor joint detection which controls communication overhead to a low level. We thoroughly examine existing techniques that may be applied, and identify their performance limitations. We then propose two new techniques, called *combinable filters* and *progressive filtering*, which address the performance limitations from different angles. We formally prove the correctness of our new solutions based on a probabilistic joint detection model. Numerical evaluation shows that our best solution achieves an overhead reduction in the range of 63% to 91% over the Bloom filter solution under various simulation settings when the number of monitors is 10 or more.**

I. Introduction

Searching for widespread events in large networked systems is a fundamental function that underlies many important applications. A number of examples are given below.
1. Consider an ISP or a large enterprise deploying a number of honeypots (monitors) in its network to detect cyber-attacks [1]. As honeypots do not provide any real service, those who contact them are possibly malicious. Each event may be characterized by a source address (possibly a contacted destination port as well). Compared to isolated events, widespread malicious acts demand immediate attention. This will require the central coordinator to communicate with the honeypots and identify in real time the common events happening in all or a subset of honeypots at critical locations (i.e., the source addresses that contact all of those honeypots).
2. Consider a distributed intrusion detection system where multiple IDSes (monitors) individually guard geographically dispersed subnets and they also cooperatively communicate with a central coordinator to detect threats of common interest, where each event may also be characterized by an address and a type of threat from that address.
3. For traffic engineering, ISPs may perform measurement on their networks to improve QoS [2]. One measurement useful for resource distribution is to find the set of flows traveling along a path. To do so, a sniffer device may be collocated with each router along the path to record the identifiers of all passing flows by examining TCP SYN packets or some UDP packets as well, if needed. Later, by calculating the intersection of the flow sets at all routers along a particular path in a particular path, we will know the set of flows that were traveling through that path. In this case, sniffers serve as monitors and events are flows which can be characterized by the standard 5-element identifiers.
4. Online search companies such as Google have servers all over the world, and it is interesting to know the common hot subjects searched across different geographic regions. A famous example is the Google FLU Trends project that uses aggregated query terms to early detect and estimate the flu activity around the world in near real-time [3]. Suppose each server (monitor) keeps a list of keywords (events) that were frequently searched recently. One can perform a join over the lists of keywords in different areas to find out their shared concerns.

In this paper, we formulate the search of widespread events in large networked system as a network primitive function, called *multi-monitor joint detection*. Multi-monitor joint detection is performed by a cooperative monitoring system that consists of a central coordinator and a number of monitors deployed at a set of vantage points. The goal is to find the common events observed by all or a given subset of monitors during each measurement period.

It is a challenging problem to search for widespread events because large-scale cooperative monitoring can generate tremendous communication overhead that overburdens the coordinator. To begin with, consider a straightforward solution, where each monitor simply reports its observed events at the end of each period. The coordinator identifies the common events from the received data, and then notifies the monitors of the common events for further action, e.g., logging all traffic related to certain events or blocking the sources that cause the events. However, this approach may incur significant network traffic, particularly, at the coordinator where data from all monitors converge. For example, consider the above application of finding the set of flows that travel through a common routing path. Suppose that there are ten millions of such flows in a measurement period of 1 minute, and each flow identifier consists of 104 bits for 5-element tuple, including source IP, source port, destination IP, destination port, and protocol. Each monitor will need to transmit $10M \times 104b = 1.04Gb$ or more flow identifiers to the central coordinator after each period. If there are 10 monitors, the coordinator will receive at least $10 \times 1.04Gb = 10.4Gb$ data every minute, which puts at least $173Mbps$ bandwidth cost on the coordinator. Moreover, it may also be a space concern for the coordinator to hold so much data and process them efficiently.

If we want to avoid sending raw data, a natural idea is for each monitor to send its set of events in a compressed form: Bloom filters easily come to mind [4], [5]. They have been heavily applied in distributed and networking systems to reduce the space and communication cost [6][7]. Each monitor encodes its event set in a Bloom filter before sending it to the coordinator. The coordinator combines the Bloom filters from the monitors into one that encodes the common events, and sends it back to the monitors. To keep the false positive low, a Bloom filter must use multiple bits to encode an event. For example, if we want to keep the false positive ratio as low as 0.001, we need 14.4 bits per event; if we want the false positive ratio to be 0.0001, we need 19.2 bits per event. In the latter case, its communication overhead is slightly less than one fifth the overhead of directly reporting all events. In the example of identifying common flows, the coordinator will still receive $10 \times 10M \times 19.2b = 1.92Gb$ data and transit a similar amount back to the monitors in each period. This overhead increases linearly as the sizes of event sets increase.

While using Bloom filters reduces the overhead considerably, this paper shows that we can improve further. By employing more sophisticated encoding and filtering techniques, we are able to further reduce the overhead by more than an order of magnitude over the Bloom filter solution in some cases. The main contributions of this paper are summarized below.

First, we give the formal model of the probabilistic multi-monitor joint detection problem in cooperative monitoring. While we do not find prior work that addresses the exact same problem, some possible solutions based on existing techniques are easy to come by. We analyze the limitation of these solutions. In particular, the combining of Bloom filters for common events will require all Bloom filters to have the same size determined by the largest event set in any monitor, which causes large overhead for a system where the event sets in the monitors vary widely.

Second, we propose *combinable filters* whose sizes are determined solely by the monitors' individual event sets, yet they can be combined to encode the common events. Using these filters, we design a new solution for multi-monitor joint detection that outperforms the Bloom filter solution, particularly when there are vastly different numbers of events at each monitor.

Third, as an equally important contribution, we propose a new filtering method called *progressive filtering*. It removes non-common events from the monitors progressively in multiple iterations. Each iteration uses a far-smaller filter whose size progressively decreases over subsequent iterations. Overall, the new method uses just $\frac{2}{\ln 2} \approx 2.9$ bits per event on average (when most elements are non-common), which compares favorably to 19.2 in the Bloom filter approach under the same false positive ratio.

Fourth, we prove the correctness of the solutions based on the problem model and evaluate their performance numerically through simulations. The results show an overhead reduction in the range of 63% to 91% by our best solution over the Bloom filter solution under various simulation settings.

The rest of this paper is organized as follows. Section II discusses the related work. Section III gives the system model and the problem statement. Section IV describes the novel solution based on the proposed combinable filters and progressive filtering techniques in detail. Section V evaluates the performance of proposed protocol by simulations. Section VI draws the conclusion.

II. Related Work

The emergence of large-scale cooperative monitoring systems has brought much research attention in recent years. Some researches focus on applying specific aggregate functions for the data collected from distributed monitors. Examples of these aggregate functions are frequency counts [8], [9], boolean predicates [10], inner products [11], [12], variance [13], and entropy [14]. For example, as a forerunner, Keralapura et al. [8] propose methods to detect the aggregate frequency of an event that is monitored by distributed nodes and exceeds a pre-defined threshold. They set dynamically-determined local threshold at each monitor, which initiates communication if the locally observed frequency count exceeds that threshold.

Meanwhile, other researches focus on tracking different distributed triggers for anomaly detection [15], [16] and latent fault detection [13]. Those studies are applied on a continuous-querying environment, which implies that the coordinator needs to continuously maintain or track the answers to queries as the monitors collect new data. More recently, monitoring distributed data streams have received much attention, with geometric monitoring approach [17], safe zone approach [18],[19], and sketching approach [20]. Cormode formalizes the continuous distributed monitoring model in his survey [21].

To the best of our knowledge, we are the first to describe and solve the probabilistic multi-monitor joint detection problem in cooperative monitoring. Directly related are Bloom filters that are applied to process database joins in distributed setting. Mackert and Lohman proposed Bloomjoin with a goal of reducing the communication cost for distributively joining two sets [22]. The idea is for one node to send a Bloom filter of its local set to filter out a large portion of unneeded data in another node before transferring unfiltered data. Mullin suggests to use partitioned Bloom filter (PBF) to encode the data, and send one segment at a time [23], where *all segments have the same* *size*. When the size of the data filtered by one segment is smaller than the size of the segment itself, it stops sending PBF segments but sends the unfiltered data instead. The advantage of this approach over Bloomjoin is that it avoids the problem of deciding on a false positive ratio as a priori for the Bloom filter construction, and its filter size is closer to the optimal. As we will show shortly, the Bloom filter approach can be extended from two sets to multiple sets (Section IV-B). However, it will run into serious performance problems, which this paper will address through new filter designs.

Michael et al. study the intersection of multiple lists [24] in a distributed setting without a central coordinator, where a series of two-set joining is performed in a linear sequence among distributed lists. Our paper studies the joining problem under a different system model with a central coordinator, allowing multi-set joining to be performed all together, instead of linearly by including one additional set at a time (which will take long time when there are numerous sites).

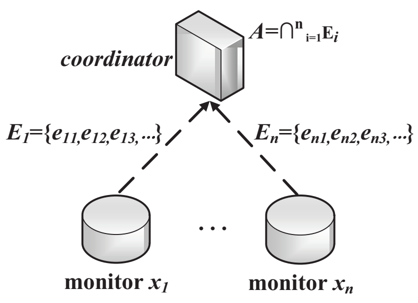
Ramesh et al. [25] propose four extensions to Bloomjoin [22], using an increment approach designed for database systems with small changes between any two consecutive join executions. This assumption does not hold in the setting of this paper. Similarly, assuming the difference between two sets is very small, the difference digest in [26] uses an invertible Bloom filter to find the set difference (i.e., elements in one set but not the other), where the number of cells in the filter is expected to be twice the size of the set difference, and each cell has a field to encode element, in addition to other fields. When most of elements in the two sets are different (a typical setting in this paper), the difference digest generates more communication overhead than simply sending the raw data.

The multi-resolution bitmaps [27] can be used for cardinality estimation, i.e., counting the number of distinct elements in a set. However, the multi-resolution bitmaps only encode the cardinalities, instead of the memberships, of the elements [28]. In other words, they do not support membership lookup. Consequently, they cannot be directly applied to solving the problem of multi-monitor joint detection that requires to check the membership of each element.

III. System Model and Problem Definition

As shown in Figure 1, consider a cooperative monitoring system deployed in a network. The system consists of a coordinator and n distributed monitors. The monitors independently capture and record distinct events, which are application-dependent. They may be events detected by honeypots, IP addresses of malicious sources identified by IDS devices, identifiers of flows passing through a router, or keywords searched on a server, as explained in the introduction. In these applications, the monitors are honeypots, IDS devices, routers and servers for web searches, respectively.

The monitors send their observed events to the coordinator according to pre-specified policy and frequency. The coordinator is responsible for collecting and synthesizing data from different monitors to form a global view. It then informs the monitors about the results and commands for followup actions.



coordinator
$A=\cap^{n}_{i=1}E_i$
$E_1=\{e_{11},e_{12},e_{13},...\}$
$E_n=\{e_{n1},e_{n2},e_{n3},...\}$
...
monitor x_1
monitor x_n

Fig. 1. Multi-monitor joint detection in Networked Cooperative Monitoring

In practice, a specific event may appear at one or multiple monitors. Consider a set of monitors of interest, $\Pi = \{x_1, x_2, ..., x_n\}$. Suppose each monitor x_i, $1 \le i \le n$, captures a set of distinct events, $E_i = \{e_{i1}, e_{i2}, e_{i3}, ...\}$. The *multi-monitor join detection problem* is to find the intersection of these sets, $A = \bigcap_{i=1}^{n} E_i$, including all common events that are observed by the monitors in Π. Depending on applications, A may be large or just a small subset of E_i. It may even be empty, in which case there is no common event for the given subset of monitors.

Finding A exactly can be expensive if the number of events in each monitor or the number of monitors is large because the coordinator has to gather a lot of information from the monitors. It is much more efficient for the coordinator to work with the monitors and help them find A approximately with a small error that is probabilistically bounded. Let $\hat{A}_i$ be the approximate set of common events that the monitor x_i has. This paper considers the *probabilistic joint detection* that has the following two requirements:

- *completeness requirement:* $A \subseteq \hat{A}_i$, and
- *accuracy requirement:* $\forall e \in E_i - A$, $Prob\{e \in \hat{A}_i\} \le \varepsilon$, where $\varepsilon \in (0, 1)$ is a small pre-defined probability value.

Namely, each monitor must find all common events, and the false positive ratio (i.e., the probability for any non-common event to be mis-classfied) is bounded by a preset value that can be made arbitrarily small. The primary performance objective is to minimize the amount of data that is received and sent by the coordinator, which is a bottleneck as it may have to communicate with a large number of monitors. Meanwhile, we will also try to reduce the communication and computation overhead of each individual monitor during the joint detection process.

Our work focuses on how to perform joint detection after each monitor has obtained its set of distinct events at the end of a measurement period. The application-dependent problem of how to obtain the event set from network traffic by a monitor with storage and processing efficiency is beyond of the scope of this paper.

IV. Multi-monitor Joint Detection

In this section, we first give a straightforward solution for comparison purpose. We then describe more efficient solutions based on Bloom filters [4][5]. We point out that the direct application of Bloom filters has two serious performance problems. Finally, we present our main technical contributions, *combinable filter* and *progressive filtering*, which together achieve significant performance gain.

A. Raw-data Solution

Suppose the coordinator has configured the set Π of monitors for joint detection of a certain type of events. At the end of a measurement period, each monitor in the subset sends its observed events to the coordinator. After receiving data from all monitors, the coordinator can find the set of common events. The coordinator can then notify each monitor of these events. The communication overhead is $\sum_{i=1}^{n} |E_i|t + n|A|t$, where t is the number of bits needed to represent each event, the first term is the total amount of data that the coordinator receives, and the second term is the total amount that it sends.

An alternative approach, referred to as *smallest first*, is to find out which monitor has the smallest event set, instruct that monitor to send its events to all other monitors which filter out non-common ones, and repeats this step such that each monitor gets a chance to send its events. The problem of this approach is that by sending the information of one monitor at a time, it takes much longer time to complete especially when the number of monitors is large. Next we will focus on the approaches that complete in one or two rounds of communications between the coordinator and the monitors. But we will also compare our new solution with the smallest first approach and show that the communication overhead of the former is considerably smaller than that of the latter in our simulations.

B. Bloom Filter Solution (BFS)

As a benchmark for comparison, we propose a Bloom filter solution, which can be considered as an extension to the existing work that performs join between two sets [22], [23]. Bloom filter is a space efficient data structure for encoding a set [4], [5]. Each filter is a bit array that is initialized to zeros, where each event is randomly mapped to k bits in the array by k hash functions, and those bits are set to ones. For membership lookup of an event b, we again map the event to k bits in the bit array and check if all of them are ones. If they are, we claim that b belongs to the set; otherwise b doesn't belong to the set.

A Bloom filter may yield *false positive* for membership lookup: a non-member event is mis-classified as a member. Let m be the number of events and l be the length of the bit array. Below we give some well-known results. The false positive ratio, i.e., the probability for any non-member to be mis-classified, under optimal setting is

$$P_{fp} = \left(\frac{1}{2}\right)^k, \qquad \text{with} \quad l = \frac{km}{\ln 2}. \tag{1}$$

Using the filter length l given above, the fraction of bits that are ones (or zeros) in such a filter is about one half. For an arbitrary non-member, each of its k bits has a chance of $\frac{1}{2}$ to be mapped to a bit of value one. The probability for all k bits to be ones (thus causing mis-classification) is $(\frac{1}{2})^k$. If we want to achieve a false positive ratio of at most ε, the values of k and l are

$$k \geq -\frac{\ln \varepsilon}{\ln 2} \tag{2}$$

$$l \geq -\frac{\ln \varepsilon}{(\ln 2)^2} m. \tag{3}$$

In the Bloom filter solution (BFS) for the joint detection problem, each monitor x_i sends the coordinator a Bloom filter BF_i that encodes its event set. The coordinator combines the received Bloom filters into a single filter by performing bitwise AND, and then sends it back to all monitors. The bitwise AND may create many zeros in the bitmap. Hence, the combined filter should be compressed before being sent out. After a monitor x_i receives the combined filter, it performs membership lookup for each event in E_i. An event is declared as a common event in $\hat{A}_i$ if its k bits in the filter are all ones. Let $P_{fp}(BF_i)$ denote the false positive ratio of BF_i.

Lemma 1: $\forall e \in A$, $\forall 1 \leq i \leq n$, $e \in \hat{A}_i$ after execution of BFS.

Proof: Event e belongs to all monitors. Its k bits are ones in all Bloom filters generated by the monitors. Those bits remain to be ones in the combined filter after bitwise AND. Hence, when x_i will find e in the combined filter and thus include it in $\hat{A}_i$. □

Lemma 2: If $\forall 1 \leq j \leq n,\ P_{fp}(BF_j) \leq \varepsilon$, then $\forall 1 \leq i \leq n,\ \forall e \in E_i - A,\ Prob\{e \in \hat{A}_i\} \leq \varepsilon$ after execution of BFS.

Proof: A non-common event e in E_i is mis-classified if all its k bits in the combined filter are ones. These bits must be ones in all BF_j, $1 \leq j \leq n$. Recall that $Prob\{e \in \hat{A}_i\}$ denotes the probability for this to happen.

Because e is not a common event, it must not belong to E_j for some $j \neq i$. $P_{fp}(BF_j)$ is the probability for the k bits of e in BF_j to be ones. Clearly, $Prob\{e \in \hat{A}_i\} \leq P_{fp}(BF_j)$.

Because $P_{fp}(BF_j) \leq \varepsilon$, $Prob\{e \in \hat{A}_i\} \leq \varepsilon$. □

In order to deal with the worse case, we shall not relax the condition of $P_{fp}(BF_j) \leq \varepsilon$. Suppose $\exists j, P_{fp}(BF_j) > \varepsilon$. Consider the case where $e \notin E_j$ but $e \in E_i$ for $1 \leq i \leq n$, $i \neq j$. The k bits of e are all ones in the combined filter if and only if they are all ones in BF_j; the probability for that to happen is $P_{fp}(BF_j)$, which is greater than ε, violating the probability requirement for mis-classification.

Theorem 1: If $\forall 1 \leq j \leq n,\ P_{fp}(BF_j) \leq \varepsilon$, BFS satisfies the completeness requirement and the accuracy requirement of probabilistic joint detection.

Proof: It follows directly from Lemmas (1)-(2) and the definitions of the two requirements in Section III. □

In order to perform bitwise AND, all Bloom filters from the monitors should have the same size l. According to (3), in order to achieve $P_{fp}(BF_j) \leq \varepsilon, \forall 1 \leq j \leq n$, the value of l should meet the following constraint:

$$l \geq -\frac{\ln \varepsilon}{(\ln 2)^2}|E_j|, \quad \forall 1 \leq j \leq n. \tag{4}$$

Let $q = \max_{j \in \{1,\ldots,n\}}\{|E_j|\}$. The minimum value of l should be

$$l = -\frac{\ln \varepsilon}{(\ln 2)^2}q, \tag{5}$$

with $k = -\frac{\ln \varepsilon}{\ln 2}$.

At the end of each measurement period, before generating the Bloom filters, the monitors need to exchange information to determine the size of their Bloom filters. Each monitor only needs to send the number of its events to the coordinator, which finds the largest number q and calculate the values of l and k before sending them back to the monitors.

For simplicity, we consider the amount of exchanged data for determining l and k to be $2n \log_2 q$ bits. The Bloom filters sent from the monitors to the coordinator and the combined one in the opposite direction have a total length of $-2n\frac{\ln \varepsilon}{(\ln 2)^2}q$ bits. Hence, the total communication overhead is $-2n\frac{\ln \varepsilon}{(\ln 2)^2}q + 2n\log_2 q$, which can be very large if ε is very small or q is very large. We stress that this is also the amount of data received/sent by the coordinator.

The compressed Bloom filters [29] may be used to reduce the communication overhead, but that will significantly increase the size of the filters prior to compression or after decompression, and thus significantly increase the memory overhead both at the monitors and at the coordinator.

Below we present our main technical contributions of combinable filters and progressive filtering. For each of them, we first point out a problem of BFS, an idea of addressing that problem, and the detailed technical description.

C. Combinable Filters

Motivation: In order to support bitwise AND, BFS requires all monitors to build Bloom filters with the same size l and the same k hash functions. The value l is set based on the size q of the largest event set at any monitor. This is non-optimal if many monitors have event sets much smaller than q. For example, in the third application example of the introduction, the heaviest-loaded router in a large network can see much more flows than lightly-loaded routers; in the fourth application example, a Google server in the US can see much more search traffic in the afternoon than a Google server in Japan where it is early Morning. It will greatly reduce the communication overhead if we allow each monitor to build its Bloom filter based on the size of its own event set.

We want to point out that the scalable Bloom filters in [30] and the incremental Bloom filters in [31] cannot solve our problem. They are designed for encoding elements from a set or multiple sets upon arrival in streaming data flows. Without the knowledge of how many elements there will be, one does not know how to set the filter size. The solution is to first use small-sized filters and add larger-sized filters on the fly when the smaller ones are filled. In this paper, we deal with a different problem. There is no need for any monitor to encode events on the fly in a Bloom filter; each monitor stores raw events. Only at the end of each measurement period, when the monitors need to send their events to the central coordinator, they generate Bloom filters to encode those events in order to cut down communication overhead. By this time, we know the exact sets to encode and hence there is no need for scalable or incremental Bloom filters [30], [31]. However, because the set sizes at different monitors may be different, their optimal filter sizes may also be different. In this case, how do we combine *variable-sized* filters by *bitwise AND* for joint detection?

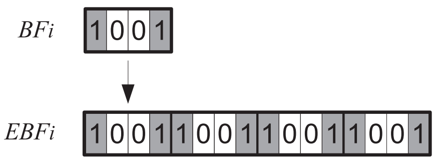


Fig. 2. Replicating BF_i to expand it for EBF_i

Below we design *combinable filters* to do just that; this will be denoted by CFS. We show how to combine variable-sized Bloom filters to retain the common events while filtering out non-common ones.

Description: Consider an arbitrary monitor x_i, $1 \leq i \leq n$. Let k and l_i be the number of hash functions and the number of bits used by BF_i, respectively. From (2) and (3), their values can be set as

$$k = -\frac{\ln \varepsilon}{\ln 2} \tag{6}$$

$$l_i \geq -\frac{\ln \varepsilon}{(\ln 2)^2}|E_i|, \tag{7}$$

in order to ensure a false positive ratio of at most ε. In order to make the filters combinable, we keep the value of k a constant across all monitors, and require the filter length to be a power of 2. Hence, we increase l_i from the lower bound, $-\frac{\ln \varepsilon}{(\ln 2)^2}|E_i|$, to the nearest power of 2 as follows:

$$l_i = 2^{\lceil \log_2(-\frac{\ln \varepsilon}{(\ln 2)^2}|E_i|)\rceil}. \tag{8}$$

For each event e in E_i, we set the bits at indices $H_j(e) \mod l_i$ in the filter to ones, where H_j, $1 \leq j \leq k$, are k independent hash functions. Clearly,

$$P_{fp}(BF_i) \leq \varepsilon, \quad \forall 1 \leq i \leq n. \tag{9}$$

Because each monitor will decide its filter size based on its own event set, there is no need for the coordinator to find the size of the largest event set, q.

After the coordinator receives the filters BF_i, $1 \leq i \leq n$, from all monitors, it expands the filters to the same size and then combine them with bitwise AND: The length of the largest filter is $2^{\lceil \log_2(-\frac{\ln \varepsilon}{(\ln 2)^2}q)\rceil}$, denoted as Q. Because the lengths of all filters are powers of 2, we can expand any filter

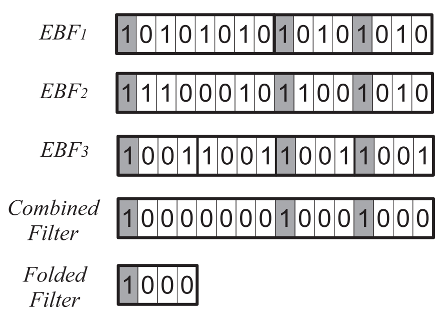


Fig. 3. Performing bitwise AND on expanded filters

BF_i with a smaller size than Q by replicating it multiple times until its length is increased to Q, as illustrated in Figure 2, where a filter of length $\frac{Q}{4}$ is replicated three times to reach the size of Q. The expanded filter is denoted as EBF_i, $1 \le i \le n$. By the nature of replication, the following lemma is true.

Lemma 3: If the jth bit in BF_i is one, then after replication, the $(j + dl_i)$th bit in EBF_i is one, $\forall 0 \le d < \frac{Q}{l_i}$.

Once all filters are expanded to the same size Q, they are combined by bitwise AND. The resulting combined filter is denoted as F, which is sent to the monitors. The AND operation creates many more zeros, as shown by the example in Figure 3. Hence, the coordinator should compress F before sending it out. Note that an optimal Bloom filter whose length is set by (1) has about half of its bits being zeros and half being ones at random locations, which offers little opportunity for compression. The lengths of the filters BF_i, $1 \le i \le n$, from the monitors to the coordinator are set larger based on (8), and thus these filters may also be compressed.

Upon receiving F, each monitor x_i performs the standard membership lookup for its events in E_i: *An event e is classified as a common event if and only if all its k bits at indices $H_j(e)$ mod Q, $1 \le j \le k$, in F are ones.*

Correctness: While the operations of filter expansion and combination are simple (which is a plus), the implication of duplicating the bits of smaller filters in bitwise AND is not obvious at all. We must formally prove that both completeness requirement and accuracy requirement will be met under such communication saving operations.

Lemma 4: For an arbitrary common event e, its k bits in the combined filter at indices $H_j(e)$ mod Q, $1 \le j \le k$, must be all ones.

Proof: Consider an arbitrary common event e. The indices of its k bits in the combined filter are $H_j(e) \mod Q, 1 \le j \le k$. To prove the bit at index $H_j(e) \mod Q$ in the combined filter is one, we need to prove that the bit at the same index in each EBF_i, $1 \le i \le n$, is one. Because Q is a multiple of l_i, we can write

$$\exists d \in [0, \frac{Q}{l_i}], H_j(e) \text{ mod } Q = dl_i + H_j(e) \text{ mod } l_i, \tag{10}$$

where d is the quotient of $H_j(e)$ mod Q divided by l_i and $H_j(e)$ mod l_i is the remainder. Because $e \in BF_i$, the bit at index $H_j(e) \mod l_i$ is one. By Lemma 3, the bit at index $H_j(e) \mod Q$ in EBF_i must be one, which completes the proof. □

Lemma 5: $\forall e \in A$, $\forall 1 \le i \le n$, $e \in \hat{A}_i$ after execution of CFS.

Proof: It follows directly from Lemma 4. □

Lemma 6: $\forall 1 \le i \le n,\ \forall e \in E_i - A,\ Prob\{e \in \hat{A}_i\} \le \varepsilon$ after execution of CFS.

Proof: Consider an arbitrary event $e \in E_i - A$. That is, $\exists i' \in [1, n]$, $i' \ne i$, $e \notin E_{i'}$. Event e is not encoded in $BF_{i'}$. However, we may still find e in $BF_{i'}$ due to false positive when the k bits of e in $BF_{i'}$ are all ones by chance. Let p be the probability of an arbitrarily chosen bit to be one in $BF_{i'}$. For the k bits of e in $BF_{i'}$, each of them has a probability of p to be one. The probability for all k bits to be ones is thus p^k, which is the false positive ratio. We know that $P_{fp}(BF_{i'}) \le \varepsilon$. Hence, $p^k \le \varepsilon$.

Consider the k bits of e in the combined filter F. Let b be an arbitrary one of them at index $H_j(e) \mod Q$, $\forall j \in [1..k]$, where Q is the length of F (also the largest length among the filters received by the coordinator from all monitors). The bit b will be one if and only if the bits at the same index in all expanded filters are ones. Hence, a necessary condition for b to be one is that the bit at the same index in $EBF_{i'}$ is one, i.e., the bit at index $H_j(e) \mod l_{i'}$ in $BF_{i'}$ is one, according to the nature of the filter expansion as defined earlier in this section. That bit is one of the k bits of e in $BF_{i'}$, and the probability for it to be one is p (see the previous paragraph).

Because the probability of satisfying a necessary condition is p, the probability for b to be one must not exceed p. Since b can be any one of k bits, the probability for all k bits to be ones (causing false positive) must not exceed p^k. Hence, the false-positive ratio of F, denoted by $Prob\{e \in \hat{A}_i\}$, must not exceed p^k, which is bounded by ε. □

Theorem 2: If $\forall 1 \le j \le n$, CFS satisfies the completeness requirement and the accuracy requirement of probabilistic joint detection.

Proof: It follows directly from Lemmas (5)-(6). □

The total communication overhead from the monitors to the coordinator is $\sum_{i=1}^{n} 2^{\lceil \log_2(-\frac{\ln \varepsilon}{(\ln 2)^2}|E_i|)\rceil}$. The total communication overhead from the coordinator to the monitor is $n \times \min_{i \in \{1,\dots,n\}}\{2^{\lceil \log_2(-\frac{\ln \varepsilon}{(\ln 2)^2}\{|E_i|\})\rceil}\}$. The delay of CFS is smaller than BFS. It takes only one round trip of communication between the monitors and the coordinator, whereas BFS takes two round trips with the first determining the value of q and the second exchanging the filters.

D. Progressive Filtering

Motivation: The accuracy requirement ε can be very small in practice, particularly when A only has a small number of common events. Suppose E_i is in millions but A is in hundreds. The number of non-common events that are misclassified into $\hat{A}_i$ is $|E_i - A|\varepsilon \approx |E_i|\varepsilon$, which can dwarf the number of common events unless we choose a small value for ε. As an example, if $\varepsilon = 10^{-5}$, $l_i = 28.6|E_i|$ from (7), which means 28.6 bits per event!

To address this inefficiency, we propose a new approach called *progressive filtering* to remove non-common events with less communication overhead: Instead of using a single filter per monitor with $k = -\frac{\ln \varepsilon}{\ln 2}$ and false positive ratio ε, we use multiple filters per monitor, each with a larger false positive ratio. For example, we may use $-\frac{\ln \varepsilon}{\ln 2}$ filters per monitor, each with $k = 1$ and false positive ratio $\frac{1}{2}$, such that the combined false positive ratio is $(\frac{1}{2})^{-\frac{\ln \varepsilon}{\ln 2}} = \varepsilon$. The reason for doing so is that as we apply one filter after another, we are able to progressively reduce the sizes of subsequent filters such that their combined size is much smaller than the size in the single filter approach.

Description: Progressive filtering is performed in iterations. Each iteration is essentially an execution of CFS where each monitor contributes a filter with $k = 1$. More specifically, from (1), the optimal filter setting is, $\forall 1 \le i \le n$,

$$l_i = \tfrac{|E_i|}{\ln 2} \tag{11}$$

$$P_{fp}(BF_i) = \tfrac{1}{2}. \tag{12}$$

In order for the filter to be combinable, it is required that l_i to be a power of 2. Hence, we have to set

$$l_i = 2^{\lceil \log_2 \frac{|E_i|}{\ln 2} \rceil} \tag{13}$$

$$P_{fp}(BF_i) \le \tfrac{1}{2}. \tag{14}$$

Applying this CFS, the completeness requirement will be met due to Lemma 5. Substituting ε by $\frac{1}{2}$ in Lemma 6, we know that the accuracy requirement is met not for $Prob\{e \in \hat{A}_i\} \le \varepsilon$, but for $Prob\{e \in \hat{A}_i\} \le \frac{1}{2}$, $\forall 1 \le i \le n$, $\forall e \in E_i - A$. Each non-common event will be identified with a probability of at least $\frac{1}{2}$. It also means that at least half of all non-common events are expected to be identified and removed from $|E_i|$. If the number of common elements is small, then the value of $|E_i|$ is about halved.

We repeat the above CFS with $k = 1$ for $-\frac{\ln \varepsilon}{\ln 2}$ times in total, using a progressively smaller filter size l_i each time due to a shrinking value of $|E_i|$. Note that a different hash function should be used each time. This solution is referred to as CFSP for Combinable Filter Solution with Progressive filtering.

Correctness: Because the completeness requirement is met after each execution of CFS with $k = 1$ (Lemma 5), all common events will be found. Because the probability for a non-common event to be identified is at least $\frac{1}{2}$ for each execution of CFS with $k = 1$, the total probability of being identified over $-\frac{\ln \varepsilon}{\ln 2}$ independent CFS executions is at least $(\frac{1}{2})^{\frac{\ln \varepsilon}{\ln 2}} = \varepsilon$. Hence, the accuracy requirement is also met.

Intuition for Communication Reduction: We know that each iteration of CFS with $k = 1$ helps all monitors x_i identify and remove half of their non-common events, and $|E_i|$ is expected to be halved when most events are non-common. To ease our augment, let's approximately use $l_i = \frac{|E_i|}{\ln 2}$ instead of (13) to make the point. As we repeat CFS iteratively for $-\frac{\ln \varepsilon}{\ln 2}$ times, with the value of $|E_i|$, $1 \le i \le n$, halved each time, the value of l_i is also halved each time because it is linear in $|E_i|$. Hence, the $(-\frac{\ln \varepsilon}{\ln 2})$ filters from each monitor x_i has a total length bounded by

$$\sum_{j=1}^{-\frac{\ln \varepsilon}{\ln 2}} \frac{|E_i|}{\ln 2} \times (\frac{1}{2})^{j-1} < \sum_{j=1}^{\infty} \frac{|E_i|}{\ln 2} \times (\frac{1}{2})^{j-1} = \frac{2}{\ln 2}|E_i|,$$

where $|E_i|$ refers to the size of the original event set at x_i before any filtering. Note that this bound is independent of the value ε. Under progressive filtering, the overhead of $\frac{2}{\ln 2}$ bits per event is much smaller than 28.6 bits per event without progressive filtering when $\varepsilon = 0.0001$.

Two Rounds of Communications Between Monitors and Coordinator: One problem of CFSP is that it takes $(-\frac{\ln \varepsilon}{\ln 2})$ rounds of communications between the monitors and the coordinator, which means that much longer delay than CFS. To reduce the number of rounds, we can set a higher value of k in each execution of CFS. In general, we can set the value of k arbitrarily as long as the sum of the k values over all executions is $(-\frac{\ln \varepsilon}{\ln 2})$. For example, to reduce the number of rounds to two (the same as what BFS needs), we may first apply CFS with $k = 1$ and then apply CFS with $k = -\frac{\ln \varepsilon}{\ln 2} - 1$. We will use this version of CFSP in our simulations in order to compare it with BFS on a fair ground in terms of delay.

V. Performance Evaluation

A. Simulation Setting

We compare the proposed CFS and CFSP with two benchmark solutions, the raw-data solution and BFS. All of them take one (CFS and raw data) or two (CFSP and BFS) rounds of communication between the monitors and the coordinator. Later we will also compare with Smallest First (Section IV-A), which takes many more rounds of communication.

The performance metrics include the communication overhead received/sent by the coordinator (which is also the overall communication overhead) and the total computation overhead measured by the number of hash operations done by all monitors. Hashing is the main overhead of encoding event sets in filters, where each event takes k hashes for a Bloom filter and one or multiple hashes for a combinable filter, while setting the bits to ones are relatively cheap. The bitwise AND at the coordinator is also trivial. We use CRC-64 as our master hash function, which can be performed very efficiently, and k hash functions can be derived by appending k different seed numbers to the input.

Although the application scenario can be any of the examples given in the introduction, we use the keyword-search application to be concrete. Each of the servers keeps a set of keywords, phrases or sentences that are searched by users in a measurement period. A query is made to find the common keywords/phrases/sentences searched on n servers. Under this setting, each server is a monitor and each event is a keyword/phrase/sentence (being searched). To be consistent with the rest of the paper, we will still use the terms of monitor and event. Suppose we hash each event to 64 bits long in the raw-date solution to avoid excessive overhead.

We define the *intersection ratio* as

$$R_{INTS} = \frac{|A|}{min_{i \in \{1,\ldots,n\}}\{|E_i|\}}, \qquad (15)$$

which is the ratio between the number of common events and the minimum size among the event sets in all monitors. If not otherwise specified, the default parameters are $R_{INTS} = 0.5$, $\varepsilon = 0.001$, and $n = 10$. We will vary each one of them in the simulations. The number of events in each monitor is randomly chosen between 100,000 and 1,000,000 by default, which means the total amount of data needed to be collected varies from 0.8GB to 8GB using the raw-data solution, but we will also change this range during evaluation. For each set of parameters, we will repeat the simulation with a random seed for 1,000 times to obtain the average result as one data point. The standard deviation of all data points is within 3%.

B. Communication Comparison w.r.t. R_{INTS} and ε

Our first set of simulations compare the four solutions in terms of communication overhead with respect to R_{INTS} and ε. The results are shown in Figures 4-6, where the horizontal axis is the value of R_{INTS} from 0 to 1 and the vertical axis is the communication overhead in unit of Mb. From Figure 4 with $\varepsilon = 0.01$, the overhead of the Bloom filter solution is about half the overhead of the raw-data solution. The overhead of CFS is again about half that of the Bloom filter solution. The overhead of CFSP is far less than half that of CFS. For example, when $R_{INTS} = 0.5$, the overheads of the raw-data solution, the Bloom filter solution, CFS and CFSP are 410.4Mb, 176.5Mb, 103.5Mb and 45.5Mb, respectively. That means 89% and 74.2% overhead saving by CFSP over the raw-data solution and the Bloom filter solution, respectively. In the figure, the overheads of the Bloom filter solution do not change with R_{INTS} because the amount of data transferred is not sensitive to the number of common events. On the contrary, CFSP progressively filters out non-common events. Hence, its filters have larger sizes when there are more common and thus fewer non-common events, which in turn means fewer events are filtered at each iteration. Nevertheless, even when R_{INTS} reaches the highest value of 1, the overhead saving by CFSP is still significant.

Figures 5-6 present results under different values of ε. The performance gain by CFS and CFSP over the Bloom filter solution remains large. Interestingly, the gain by the Bloom filter solution over the raw-data solution diminishes as ε becomes very small. The reason is that the amount of data transferred by the raw-data solution does not depend on the value of ε, whereas the size of a Bloom filter is linear in $\ln \varepsilon$. In fact, if ε is sufficiently small, the size of a Bloom filter will surpass the raw data that it encodes.

C. Communication Comparison w.r.t. Number of Monitors

Our second set of simulations compare the four solutions in terms of communication overhead with respect to n, the number of monitors. The results are shown in Figure 7, where the horizontal axis is the number of monitors from 2 to 50. We can see that the overhead of all four solutions increases linearly with the number of monitors. The relative gaps remain largely constants. In absolute values, the performance gains by the proposed solutions increase when there are more monitors. For example, when $n = 20$, the overheads of the raw-data solution, the Bloom filter solution, CFS and CFSP are 702.6Mb, 547.8Mb, 279.5Mb and 55.7Mb; respectively. When

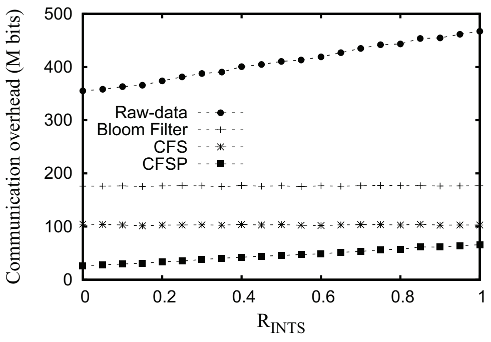


Fig. 4. Performance comparison of four solutions in terms of communication overhead with respect to R_{INTS} when $\varepsilon = 0.01$

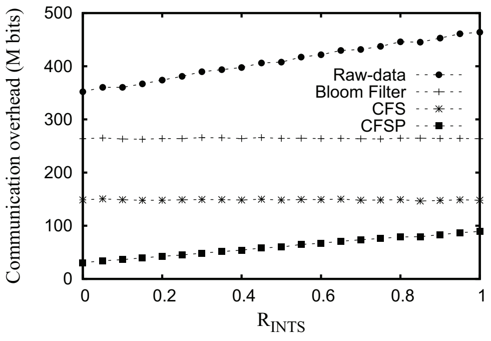


Fig. 5. Performance comparison of four solutions in terms of communication overhead with respect to R_{INTS} when $\varepsilon = 0.001$

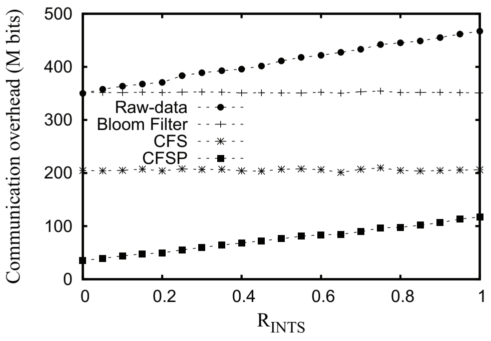


Fig. 6. Performance comparison of four solutions in terms of communication overhead with respect to R_{INTS} when $\varepsilon = 0.0001$

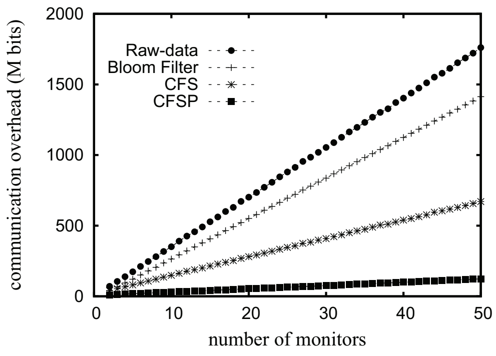


Fig. 7. Performance comparison in terms of communication overhead with respect to the number of monitors from 2 to 50 under $\varepsilon = 0.001$ and $R_{INTS} = 0.5$

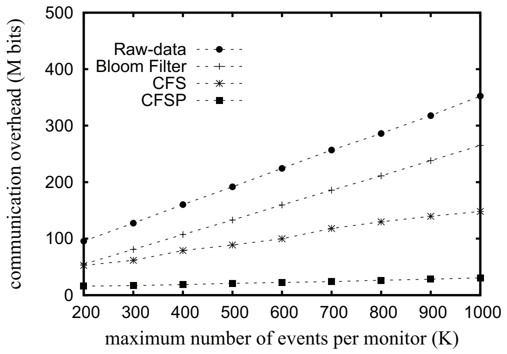


Fig. 8. Performance comparison in terms of communication overhead under $\varepsilon = 0.001$, $R_{INTS} = 0.5$, and $n = 10$. The horizontal axis is the maximum number of events per monitor. The unit on the horizontal axis is K (one thousand). The minimum number is the default value of 1. The actual number of events for each monitor is randomly picked from the range between the minimum and the maximum.

$n = 40$, the overheads of the raw-data solution, the Bloom filter solution, CFS and CFSP are 1410.4Mb, 1125.1Mb, 542.5Mb and 99.7Mb, respectively.

D. Communication Comparison w.r.t. Range of Event Sets

Our third set of simulations compare the four solutions in terms of communication overhead with respect to the range of event-set size. The results are shown in Figure 8, where the horizontal axis shows the maximum number of events per monitor and the unit is K (one thousand). The actual number of events in each monitor is randomly chosen from a range from 100,000 to the maximum number. The overheads of all solutions increase linearly with this range because each filter has to encode proportionally more events. The performance gains by the proposed solutions increase when the maximum number of events increases. For example, the performance gains are 71%, 86%, 89% when the maximum number of events are 200, 500, 1000, respectively. The reason is that when the differences between each monitor are larger, the wasted overhead of the Bloom filter solution is larger as all Bloom filters sent by the monitors have the same size.

TABLE I
COMPUTATION OVERHEAD OF THREE SOLUTIONS WITH RESPECT TO R_{INTS} WHEN $\varepsilon = 0.01$. THE UNIT IS M (ONE MILLION). FOR EXAMPLE, 38.2 IN THE TABLE MEANS 38.2 MILLIONS.

R_{INTS}	average number of hash operations per monitor					
	0.00	0.20	0.40	0.60	0.80	1.00
Bloom Filter	38.2	38.2	38.4	38.4	38.4	38.5
CFS	38.2	38.2	38.4	38.4	38.4	38.5
CFSP	7.6	9.3	11.2	12.9	14.4	16.3

TABLE II
COMPUTATION OVERHEAD OF THREE SOLUTIONS WITH RESPECT TO R_{INTS} WHEN $\varepsilon = 0.001$. THE UNIT IS M (ONE MILLION).

R_{INTS}	average number of hash operations per monitor					
	0.00	0.20	0.40	0.60	0.80	1.00
Bloom Filter	54.7	54.7	54.8	54.8	54.9	55.0
CFS	54.7	54.7	54.8	54.8	54.9	55.0
CFSP	8.7	11.3	13.9	16.4	19.0	21.5

E. Computation Comparison

Our fourth set of simulations evaluate the computation overhead under the same parameter settings as the first set of simulations. The results are shown in Tables I-III. As the raw-data solution does not have hash operations, it is not included in the tables. The computation overheads of the Bloom filter solution and CFS are the same because they encode the same number of events in each filter with the same number of hashes. Their computation overheads are not sensitive to the number of common events. The computation overhead of CFSP is much smaller because of progressive filtering. The overhead decreases as the number of common events decreases. For example, when $\varepsilon = 0.01$, the saving computation overhead by CFSP over the Bloom filter solution increases from 57% to 80.1% when we vary R_{INTS} from 1 to 0. The reason is that when there are more non-common events, more are filtered in each iteration and less are encoded, resulting in smaller computation overhead.

We use a desktop computer with a 2.0 GHz x86_64 CPU and 8G RAM to run the code of the coordinator and compare the computation times of the four solutions. We set $\varepsilon = 0.001$, $R_{INTS} = 0.5$, and $n = 50$. The number of events observed by each monitor is randomly picked from the range between 0 and a given maximum number varying from 200,000 to 1,000,000. We repeat the simulation 1,000 times under each setting and calculate the average computation time of the coordinator. The results are shown in Figure 9. We can see that the computation time of all solutions increases linearly with the increase of the maximum number of events since the coordinator needs to

TABLE III
COMPUTATION OVERHEAD OF THREE SOLUTIONS WITH RESPECT TO R_{INTS} WHEN $\varepsilon = 0.0001$. THE UNIT IS M (ONE MILLION).

R_{INTS}	average number of hash operations per monitor					
	0.00	0.20	0.40	0.60	0.80	1.00
Bloom Filter	76.8	76.8	77.2	77.3	77.5	77.0
CFS	76.8	76.8	77.2	77.3	77.5	77.0
CFSP	10.2	14.2	17.8	21.4	25.4	29.0

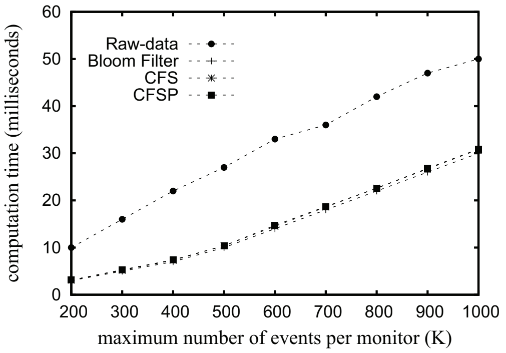


Fig. 9. Performance comparison in terms of computation time under $\varepsilon = 0.001$, $R_{INTS} = 0.5$, and $n = 50$. The horizontal axis is the maximum number of events per monitor. The unit on the horizontal axis is K (one thousand). The minimum number of events is the default value of 0. The actual number of events for each monitor is randomly picked from the range between the minimum and the maximum.

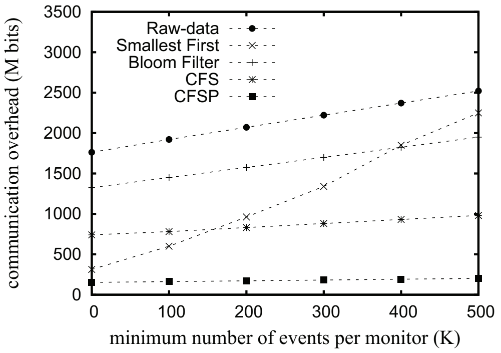


Fig. 10. Performance comparison with the smallest first solution in terms of communication overhead. under $\varepsilon = 0.001$, $R_{INTS} = 0.5$, and $n = 50$. The horizontal axis is the minimum number of events per monitor. The unit on the horizontal axis is K (one thousand). The maximum number is the default value of 1,000,000. The actual number of events for each monitor is randomly picked from the range between the minimum and the maximum.

process more data. Among the four solutions, BF takes the least computation time since the coordinator only needs to combine all Bloom filters via a simple bitwise AND operation. The computation time of CFS and that of CFSP are very close (their lines in Fig. 9 are overlapped), and they spend a little more time on expanding the combinable filters than the BF. The computation time of the three filter-based solutions is approximately 3ms, 7ms, 14ms, 22ms and 30ms when the maximum number of events are 200, 400, 600, 800, and 1000, respectively. In contrast, the raw-data solution requires much more computation time because it has to calculate the intersection of all sets of events. More specifically, its computation time is 11ms, 22ms, 33ms, 42ms and 50ms when the maximum number of events is set to 200, 400, 600, 800, and 1000, respectively. The experimental results demonstrate that the computation time of the coordinator is very small for all solutions.

F. Comparison with Smallest First

Our fifth set of simulations compare CFS, CFSP, BFS and the raw-data solution with the smallest first solution when the number of monitors is 50. For the smallest first solution, when a monitor wants to send its events to others, it does so through the coordinator, so that the monitors do not have to know each other, which is the case for other solutions. The time comparison is given as follows. CFS and the raw-data solution takes one round of communication between the monitors and the coordinator. CFSP and BFS take two rounds. The smallest first solution takes 100 rounds.

The communication overhead comparison is given in Figure 10, where the horizontal axis shows the minimum number of events per monitor in unit of 1000. The actual number of events in each monitor is randomly chosen from a range between the minimum and 1,000,000. The overheads of all solutions increase linearly as the average number of events per monitor increases because either the monitors have to send more events or their filters have to encode more events. When the average number of events per monitor is very low, the overhead of the smallest first solution is also very low, though still higher than CFSP. As the number of events increase, the slope of overhead increase in smallest first is much larger than that in CFSP. Consequently, the gap between them widens quickly. We want to point out that the overhead comparison with small event numbers is less important and the comparison with larger event numbers is critical because that is when the coordinator may be under communication stress.

G. Special Cases

One special case is that the common set is empty, i.e., the intersection ratio $R_{INTS} = 0$. From Figure 4 with $\varepsilon = 0.01$, when $R_{INTS} = 0$, the overhead of CFSP is about one fourth, one sixth and one fourteen as the overhead of CFS, BFS and the raw-data solution, respectively. The performance gain becomes larger with ε becomes smaller as shown in Figure 5 and Figure 6. Actually, the performance gain is largest when $R_{INTS} = 0$. The reason is the size of the combined filter in the first round will be very small when the intersection of all subsets is empty. Then the subsequent communication overhead will be small. Hence, CFSP is also the best solution when the common set is empty.

Another special case is one or more subsets are exactly the same as the common set, i.e., the intersection ratio $R_{INTS} = 1$. From Figure 4 with $\varepsilon = 0.01$, when $R_{INTS} = 1$, the overhead of CFSP is about 75%, 40% and 15% as the overhead of CFS, BFS and the raw-data solution, respectively. CFSP is also the best solution as shown in Figure 5 and Figure 6.

VI. Conclusion and Future Work

This paper defines a formal problem model for probabilistic joint detection in cooperative monitoring, which is to find the events that appear in every monitor with a user-specified accuracy. We propose two new techniques to solve the joint detection problem with much reduced overhead. The first technique is called combinable filters. Although these filters have variable lengths based on their individual event sets encoded, they can be combined for common events. The second technique is called progressive filtering, which removes non-common events with small, progressively diminishing filters. Our simulations show that the new solutions consistently outperform Bloom filters and are able to reduce communication overhead by an order of magnitude sometimes.

Our future work is to investigate the joint detection problem in a purely distributed setting without the central coordinator. The communication will be performed amongst the monitors in P2P [32] or other means. The overall communication overhead is likely to be higher in a fully distributed environment. Although the two new techniques proposed in this paper may still be applicable, new challenges arise on how to coordinate the information flows among the monitors in order to minimize the amount of data to be exchanged. Another research direction is to integrate techniques of widespread event detection into other systems (such as contention distribution [33] and security design [34]) to make the latter's performance adaptable to dynamic conditions.

VII. Acknowledgments

This work was supported in part by the National Science Foundation of China under grant Nos. 61379145, 61170288, 61202429 and US National Science Foundation under grant NeTS 1115548.

References

[1] L. Spitzner, "The honeynet project: Trapping the hackers," *IEEE Security & Privacy*, vol. 1, no. 2, pp. 15–23, 2003.

[2] T. Li, S. Chen, and Y. Qiao, "Origin-destination flow measurement in high-speed networks," *Proc. of INFOCOM*, pp. 2526–2530, 2012.

[3] Jeremy Ginsberg, Matthew Mohebbi, Rajan Patel, Lynnette Brammer, Mark Smolinski, and Larry Brilliant, "Detecting influenza epidemics using search engine query data," *Nature*, vol. 457, pp. 1012–1014, 2009, doi:10.1038/nature07634.

[4] 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.

[5] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2003.

[6] S. Tarkoma, C.E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *Communications Surveys Tutorials, IEEE*, vol. 14, no. 1, pp. 131–155, First 2012.

[7] Yan Qiao, Tao Li, and Shigang Chen, "Fast bloom filters and their generalization," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 1, pp. 93–103, Jan 2014.

[8] R. Keralapura, G. Cormode, and J. Ramamirtham, "Communication efficient distributed monitoring of thresholded counts," *Proc. SIGMOD*, pp. 289–300, 2006.

[9] C. Olston, J. Jiang, and J. Widom, "Adaptive filters for continuous queries over distributed data streams," *Proc. SIGMOD*, pp. 563–574, 2003.

[10] S. Agrawal, S. Deb, K. V. M. Naidu, and R. Rastogi, "Efficient detection of distributed constraint violations," *Proc. ICDE*, p. 1320C1324, 2007.

[11] G. Cormode and M. N. Garofalakis, "Approximate continuous querying over distributed streams," *ACM Transactions on Database Systems*, vol. 33, no. 2, 2008.

[12] A. Friedman, I. Sharfman, D. Keren, and A. Schuster, "Privacy-preserving distributed stream monitoring," *Proc. NDSS*, 2014.

[13] M. Gabel, A. Schuster, and D. Keren, "Communication-efficient distributed variance monitoring and outlier detection for multivariate time series," *Proc. IPDPS*, 2014.

[14] C. Arackaparambil, J. Brody, and A. Chakrabarti, "Functional monitoring without monotonicity," *Proc. ICALP*, pp. 95–106, 2009.

[15] Ling Huang, Minos Garofalakis, A.D. Joseph, and N. Taft, "Communication-efficient tracking of distributed cumulative triggers," in *Distributed Computing Systems, 2007. ICDCS '07. 27th International Conference on*, June 2007, pp. 54–54.

[16] L. Huang, X. L. Nguyen, M. Garofalakis, J. M. Hellerstein, M. I. Jordan, A. D. Joseph, and N. Taft, "Communication-efficient online detection of network-wide anomalies," *Proc. INFOCOM*, 2007.

[17] I. Sharfman, A. Schuster, and D. Keren, "A geometric approach to monitoring threshold functions over distributed data streams," *TODS*, 2007.

[18] D. Keren, I. Sharfman, A. Schuster, and A. Livne, "Shape sensitive geometric monitoring," *Trans. on Knowl. and Data Eng.*, 2012.

[19] A. S. Izchak Sharfman and D. Keren, "Shape sensitive geometric monitoring," *Proc. PODS*, 2008.

[20] G. Cormode and M. Garofalakis, "Sketching probabilistic data streams," *Proc. SIGMOD*, 2007.

[21] G. Cormode, "Distributed top-k monitoring," *SIGMOD Rec.*, 2013.

[22] Lothar F. Mackert and Guy M. Lohman, "R* optimizer validation and performance evaluation for local queries," *SIGMOD Rec.*, vol. 15, no. 2, pp. 84–95, June 1986.

[23] J. K. Mullin, "Optimal semijoins for distributed database systems," *IEEE Trans. Softw. Eng.*, vol. 16, no. 5, pp. 558–560, May 1990.

[24] L. Michael, W. Nejdl, Odysseas Papapetrou, and Wolf Siberski, "Improving distributed join efficiency with extended bloom filter operations," in *Advanced Information Networking and Applications, 2007. AINA '07. 21st International Conference on*, May 2007, pp. 187–194.

[25] Sukriti Ramesh, Odysseas Papapetrou, and Wolf Siberski, "Optimizing distributed joins with bloom filters," in *Proceedings of the 5th International Conference on Distributed Computing and Internet Technology*, Berlin, Heidelberg, 2008, ICDCIT '08, pp. 145–156, Springer-Verlag.

[26] David Eppstein, Michael T. Goodrich, Frank Uyeda, and George Varghese, "What's the difference?: Efficient set reconciliation without prior context," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 218–229, Aug. 2011.

[27] Cristian Estan, George Varghese, and Michael Fisk, "Bitmap algorithms for counting active flows on high-speed links," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 925–937, Oct. 2006.

[28] Qingjun Xiao, Shigang Chen, Min Chen, and Yibei Ling, "Hyper-compact virtual estimators for big network data based on register sharing," in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, New York, NY, USA, 2015, SIGMETRICS '15, pp. 417–428, ACM.

[29] Michael Mitzenmacher, "Compressed bloom filters," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 604–612, Oct. 2002.

[30] Paulo Sérgio Almeida, Carlos Baquero, Nuno Preguiça, and David Hutchison, "Scalable bloom filters," *Inf. Process. Lett.*, vol. 101, no. 6, pp. 255–261, Mar. 2007.

[31] Fang Hao, M. Kodialam, and T.V. Lakshman, "Incremental bloom filters," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, April 2008, pp. 1741–1749.

[32] Z. Zhang, S. Chen, Y. Ling, and R. Chow, "Capacity-Aware Multicast Algorithms on Heterogeneous Overlay Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 2, pp. 135–147, 2006.

[33] Y. Xia, S. Chen, C. Cho, and V. Korgaonkar, "Algorithms and Performance of Load Balancing with Multiple Hash Functions in Massive Content Distribution," *Computer Networks*, vol. 53, no. 1, pp. 110–125, June 2009.

[34] M. Yoon, S. Chen, and Z. Zhang, "Minimizing the Maximum Firewall Rule Set in a Network with Multiple Firewalls," *IEEE Transactions on Computers*, vol. 59, no. 2, pp. 218–230, February 2010.