2014 IEEE 22nd International Conference on Network Protocols

Estimating the Persistent Spreads in High-speed Networks

Qingjun Xiao[†‡] Yan Qiao[‡] Mo Zhen[‡] Shigang Chen[‡]
[†] Key Lab of Computer Network & Information Integration (Southeast Univ. of China), Education Ministry of China
[‡] Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL, USA
Emails: csqjxiao@seu.edu.cn, {yqiao, zmo, sgchen}@cise.ufl.edu

***Abstract*—The persistent spread of a destination host is the number of distinct sources that have contacted it persistently in predefined t measurement periods. A persistent spread estimator is a software/hardware component on a router that inspects the arrival packets and estimates the persistent spread of each destination. This is a new primitive for network measurement that can be used to detect long-term stealthy malicious activities, which cannot be recognized by the traditional superspreader detectors that are designed only for "elephant" activities. However, the challenge is to function such an estimator in fast but small memory space (such as on-chip SRAM of line cards), in order to keep up with the high speed of switching fabric for packet forwarding. This paper presents an implementation that can use very tight memory space to deliver high estimation accuracy: Its memory expense is less than one bit per flow element in each time period; Its estimation accuracy is over 90% better than a continuous variant of Flajolet-Martin sketches; Its operating range to produce effective measurements is hundreds of times broader than the traditional bitmap. These advantages originate from a new data structure called multi-virtual bitmap, which is designed to estimate the cardinality of the intersection of an arbitrary number of sets. We have verified the effectiveness of our new estimator using the real network traffic traces from CAIDA.**

***Keywords*—Network Traffic Measurement, Network Security, Persistent Spread Estimation.**

I. Introduction

Traffic measurement and classification in high-speed networks have many challenging problems [1], [2], [3], [4], [5], [6], [7], [8]. In this paper, we study a new problem called *persistent spread estimation*, which measures the number of distinct elements that persist in a traffic flow for a predefined number of time periods t.

As a motivation, we firstly introduce the concept of "flow spread". If treating all the packets sent towards a common destination IP address as a flow, we have a per-destination flow. We may also define a per-source flow as a stream of packets sent from a common source IP. A flow may also be TCP flows, P2P flows or other application-specific flows. For one flow, its "spread" is the number of distinct elements in its packet stream for one measurement period [7]. Here, "elements" may be destination addresses, source addresses, ports, or even keywords that appear in the packets. For example, for a per-destination flow, if we treat the source addresses in the packet stream as elements, then the flow's spread is the number of distinct source addresses that have contacted the destination IP.

The traditional *superspreader* detector is designed to identify the "elephant" flows whose spreads are abnormally large, and can be applied to monitoring network anomalies [1], [2]. For instance, if there is a spread estimator that measures the number of distinct destinations for each per-source flow, then it can be used to detect network scanners (or infected hosts), which have probed a large number of different destinations. Another example is the spread estimator of per-destination flows, which can be applied to detecting DDoS attack, in which a malicious party uses an army of compromised hosts to overwhelm a destination server.

However, the superspreader detector may fail to discover malicious activities, if attackers suppress their traffic volumes and spreads deliberately to escape the detection. We present two examples. The first is the stealthy DDoS attack, whose objective is not to overwhelm the target server by excessive external requests, but to degrade its performance using a smaller number of attacking machines. Since the number of attackers is reduced to the scale of the number of legitimate users, the gateway router cannot differentiate between the two situations of "too many users" and "under attack". The second failure case is the stealthy address/port scanning, which intentionally reduces its probing rate to avoid the detection as superspreaders. Even with a reduced probing rate, after enough time passes, the attacker can discover the system vulnerabilities he may exploit. Or more deadly, he can use an army of compromised nodes to perform coordinated low-rate scan and increase the overall probing speed [9]. In summary, for all the examples, the stealthy attacks exhibit a common traffic pattern: There are a small set of source (destination) nodes that contact a destination (source) node persistently for a long period.

We propose to detect the low-rate stealthy attackers by measuring the persistent traffic they generate. This is possible since their long-term activities strongly differ from the short-term traffic generated by legitimate users. According to our analysis of real-world network traces from CAIDA (Cooperative Association for Internet Data Analysis) [10], the continuous interaction between legitimate users and their target HTTP/HTTPS servers is normally shorter than twenty minutes. In contrast, the traffic flows of the stealthy attacks demonstrate a dramatically different pattern — in a per-destination or per-source flow, there are an abnormally large number of persistent elements (that stay in the flow for at least t time periods). An

978-1-4799-6204-4/14 $31.00 © 2014 IEEE
DOI 10.1109/ICNP.2014.33

example is the stealthy DDoS attack. Since its objective is to degrade the performance of target server in a desired long period, it inevitably involves a certain number of machines sending requests persistently to a target server, demonstrating a traffic pattern that a large number of persistent elements exist in a per-destination flow. Another example is the stealthy network scan. Since the attacker wants to improve the probing efficiency, he intentionally avoids the network segment scanned in one period to overlap with the segment in another. Any source node that avoids the overlapping for a long enough time is probably an attacker that wants to probe the network. This corresponds to a traffic pattern that a per-source flow contains a negligibly small amount of persistent elements.

When implementing a persistent spread estimator, the key challenge is to fit it into a small high-speed memory. Today's core routers forward most packets on the fast forwarding path between network interface cards that bypasses CPU and main memory. To keep up with such high speed of line cards for packet forwarding, it is desirable to operate the estimator in the fast but expensive, size-limited on-chip SRAM [2]. Considering that many other essential routing/security/performance functions may also run from SRAM, it is critical to design the estimator's data structure as compact as possible. Moreover, it is important for the estimator to use the tight space to support a large operating range, in order to produce effective measurements for elephant flows. In real networks, the persistent spreads of flows are distributed in an extremely imbalance manner: The persistent spread of some flows are likely to be extraordinarily larger than the rest, which are called elephant.

In this paper, we propose an implementation of the persistent spread estimator based on a data structure called *multi-virtual bitmaps*. The size of on-chip SRAM space it requires does not relate with the number of time periods t, but depends on the number of flow elements that pass through a router in only one time period. More precisely, in each time period, its required size of SRAM is less than one bit per flow element.

Even given such limited space, our algorithm is able to deliver high estimation accuracy. The evaluation results show that our estimator is 90% more accurate than a continuous variant of Flajolet-Martin sketches [5]. Such an improvement comes from our observation that in real network traffic traces, the continuous interaction of legitimate users with a web server is pretty short in time duration, which is typically less than twenty minutes. Hence, it is possible to filter the traffic from the short-term behaviors of legitimate users, and retain the long-term persistent traffic which may link with stealthy DDoS attacks or network scanning. Moreover, the estimation accuracy of our algorithm improves as the number of measurement periods t grows, because it is able to filter the short-term traffic of legitimate users more effectively. The accuracy gain as t grows is a useful feature that allows a network administrator to increase t arbitrarily to distinguish persistent elements from normal transient traffic.

Our algorithm named multi-virtual bitmaps provides another advantage that extends the operating range of producing effective measurements by hundreds of times, as compared with the traditional bitmap method, which allocates each flow with an equal-sized and separated bitmap. In contrast, our method allows different flows to share bits from a common bit pool. By drawing bits randomly from the pool, an individual flow constructs a virtual bitmap, for the estimation of its persistent spread. Through bit sharing, large flows can "borrow" bits from small flows to extend their effective operating range. We have evaluated the performance of our proposed algorithm, including memory expense, estimation accuracy and operating range, by experiments based on real network traffic traces.

The rest of this paper is organized as follows. Section II formulates the problem of persistent spread estimation. Section III presents two naive solutions to motivate our algorithm, which is elaborated in Section IV. In Section V, we analyzes its bias and variance. Section VI enhances our solution based on multi-virtual bitmaps, in order to expand the operating range to deal with large flows. Section VII evaluates our proposed algorithms by experimental results. Section VIII describes the related work. Section IX draws the conclusion.

II. Problem Definition

In this section, we formalize our research problem. A *persistent spread estimator* is a software/hardware module on a gateway router (or a core router) to monitor the traffic flows passing through the router. Here, a flow can be either a per-destination flow or a per-source flow. An example of a per-destination flow is illustrated in Fig. 1, where a server inside an intranet is contacted by a set of external hosts. All the packets sent from the external hosts to the server constitute a per-destination flow, which is inspected by the gateway.

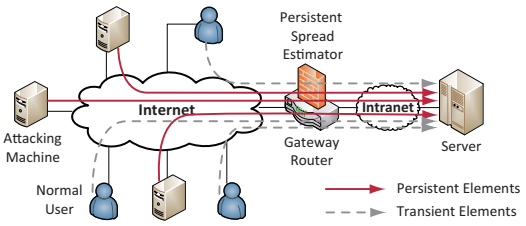


Fig. 1: Persistent spreads can help detect stealthy DDoS attacks.

A per-destination (source) flow is all the packets towards a common destination (source) address, and the flow elements are the source (destination) addresses in the packet stream. For a flow of interest, let S_i be the set of flow elements observed by the router in the ith measurement period. These elements can be divided into two subsets. (1) The elements in the set $S^* = S_1 \cap S_2 \ldots \cap S_t$ are called the *persistent elements*, which stay in the flow for t consecutive periods, and t is a system parameter that is configurable by network administrators. (2) The elements in the set $S_i - S^*$ are called the *transient elements* in the ith period. Typically, for a transient element, its packets can be observed by the gateway router only in a few periods, which is similar to a normal user finishing his/her online transaction within one or two periods. We do not deny the possibility that a small proportion of users are heavy users that occupy more than two periods. We only need their online time to be smaller than the number of periods t, which makes them differ from the persistent elements.

Our paper is to design an algorithm that can efficiently estimate the cardinality of persistent elements $|S^*|$, or called persistent spread, for a flow of interest. This problem has many important applications, and we list just a few. (1) For per-destination flows, their persistent spreads can help to detect

the stealthy DDoS attacks or the forging of server popularity. An example of stealthy DDoS attacks is illustrated in Fig. 1, where three attacking machines send requests repeatedly to the server to downgrade its performance. We want to detect the existence of these persistent attacking hosts from their number. (2) For per-source flows, their persistent spreads can help detect network scanning. Since a network scanner avoids the redundant probing of the same network segment, its persistent spread $|S^*|$ is ultra low, while the number of destination addresses $|S_i|$ it have contacted in each period is considerable.

For simplicity, we have assumed t consecutive time periods and treat their intersection $S^* = S_1 \cap S_2 \ldots \cap S_t$ as persistent elements. Note that a router can also record the traffic in non-consecutive periods and define their intersection (e.g., $S_1 \cap S_3 \ldots \cap S_{2t-1}$) as persistent elements, so that the attackers can not predict the pattern how we detect malicious activities.

A precondition for persistent spreads to be useful is that they are small in normal traffics, so that an abnormally large measurement becomes an effective indication of stealthy attacks. We verify this assumption by analyzing the real network traffic traces downloaded from CAIDA [10]. These traces are collected on January 17th, 2013 from a high-speed monitor named equinix-sanjose, connected to a 10G Ethernet backbone link. In these traces, we locate tens of server machines with large spreads in single time period. In Table I, we list the traffic pattern of four such servers which are not under attacks. In the table, the length of one measurement period is configured to ten minutes, and the number of periods t varies from one to six. We investigate the impact of t on the persistent spread $|S^*| = |S_1 \cap S_2 \ldots \cap S_t|$, when there are no persistent attacks.

TABLE I: Persistent spread decreases rapidly as the number of periods grows, and each period lasts for ten minutes.

HTTP Server 224.243.38.27/80

Number of Periods	1	2	3	4	5	6
Persistent Spread	31255	3902	223	66	30	14

HTTP Server 50.13.250.2/80

Number of Periods	1	2	3	4	5	6
Persistent Spread	50003	578	100	37	16	7

HTTPS Server 224.243.38.27/443

Number of Periods	1	2	3	4	5	6
Persistent Spread	58478	6379	780	378	227	133

HTTPS Server 224.243.38.7/443

Number of Periods	1	2	3	4	5	6
Persistent Spread	55355	8616	1661	685	301	142

This table shows that, in normal traffic with no attacks, the persistent spread reduces rapidly as t increases, and it becomes negligibly small when t is at least three. Take the HTTP server 224.243.38.27/80 as an example. When the number of periods grows to six (about an hour), the persistent spread decreases to 14, which can be neglected if compared with the one-period spread 31255. Such a phenomenon is not difficult to understand, since very few persons would keep browsing the same website for an hour without taking a break or switching to another website. For HTTPS server 224.243.38.7/443, a similar phenomenon can be found. When the number of periods is six, the persistent spread is 142, a larger number than HTTP servers. It shows that users are prone to stay online for longer time if using HTTPS as the communication protocol. But 142 is still a negligibly small amount if compared with 55355 — the spreads in one period.

We have tested other types of servers that use unfamiliar ports. We find that when contacting chat or video servers, users will stay much longer than on web servers. But the continuous online time of legitimate users is still limited for enjoying such services. Hence, if a measurement period is long enough, it can contain a typical user's continuous behavior. Since the probability for a legitimate user's online time being significantly longer than the rest is negligibly small, when the number of measurement periods t is sufficiently large, the persistent spreads lasting for the t periods become negligible. Therefore, an abnormally large persistent spread is quite likely to be a good indicator for stealthy attacks.

Without loss of generality, we consider a per-destination flow corresponding to the server dst, and we want to estimate its persistent spread $n^* = |S^*|$. We can alternate the role of source and destination, and use the same estimator to measure the persistent spread of a given source.

When implementing an estimator of persistent spreads, a challenge is to operate on the fast on-chip SRAM of routers, in order to keep up with the high speed of line cards in forwarding packets. However, the on-chip SRAM of a line card is size-limited (tens of megabytes typically), and it needs to be shared by other critical functions — routing/scheduling/security/traffic measurement. In this paper, we assume only one mega bits of on-chip SRAM are allocated for persistent spread estimation.

III. Motivation and Preliminaries

This section presents two straightforward solutions, to motivate our design based on bitwise AND of multiple bitmaps.

A. Hash Table Solutions

A naive solution is that a router records the set of source addresses S_i, $1 \le i \le t$, that have contacted the server dst in each time period. The set S_i in the ith period can be stored in the on-chip SRAM as a hash table. When the period ends, S_i can be downloaded to main memory for post-processing. With a series of such sets S_1, S_2, ..., S_t in main memory, we can calculate their intersection, which contains the persistent elements that last for t periods. The advantage of this solution is the precise calculation of persistent spread $|S_1 \cap S_2 \ldots \cap S_t|$.

However, the solution has the shortcoming of high memory cost. In the ith period, hash set S_i is kept in on-chip SRAM; when the period ends, S_i is offloaded to main memory. In this paper, we only consider the cost of precious on-chip SRAM, which for this algorithm is $O\big((32 + 32) \cdot \max(|S_i|)\big)$ bits, where the first 32 means the length of an IPv4 address for one flow element, the second 32 means the 32-bit pointer needed by the chained hash table for each element, and $\max(|S_i|)$ is the largest spread in each time period. Therefore, the memory cost is 64 bits per flow element, which is quite expensive.

In most cases, the exact values of persistent spreads are not necessary, and their approximated values with bounded estimation errors can suffice the requirement of traffic measurement. To approximate a persistent spread, a method is to store the short signatures of IP addresses into a hash table. For each IP

address x, its signature is a hash value $H(x)$ that is just k bits long ($k < 32$). This reduces memory cost by multiple folds as compared with the 32-bits IPv4 or 128-bits IPv6 addresses.

However, the enhancement by partial signatures owns two inadequacies. Firstly, it is prone to underestimate the persistent spreads: When two persistent elements are mapped to the same hash bucket and are encoded by the same signature, they will counted as one element. Secondly, its memory cost is still $O\big((k+32)\cdot\max(|S_i|)\big)$, where k is the length of a partial signature which can be 4 or 8 bits, and 32 is the length of a pointer needed by chained hash table. Hence, the memory cost is still $k+32$ bits per flow element in one period. Our vision is to reduce memory cost to less than one bit per element, and with such limited space, still render satisfactory accuracy.

B. Bitmap-based Method

We propose to adopt *bitmap* algorithm [11] for persistent spread estimation. Let B be a bit array allocated for the flow dst, called a bitmap. Its ith bit is denoted by $B[i]$, $0 \le i < m$. Its number of bits m is configured on the scale of $\max(|S_i|)$.

- At the beginning of the ith period, all the bits of array B are initialized to zero. When the router receives a packet $\langle src, dst\rangle$ that is destined to the server dst, it categorizes the packet to the flow dst, and maps the source address src to the flow's bitmap B to record the flow element. The hash function $H(src)$ decides which bit will be set in B.

$$B[H(src) \bmod m] := 1 \tag{1}$$

  Here, $:=$ is the assignment operator, and the hash function H is implemented by MurMur3 hashing. Note that this bitmap structure is "duplicate-insensitive", i.e., duplicated addresses will set the same bit and be filtered.

- At the end of the ith period, the router has recorded in the bitmap B all the source addresses that have contacted the destination server dst within this interval. We denote the bitmap of the ith period by B_i. The router will download B_i from on-chip SRAM to DRAM for post-processing.

Given a sequence of bitmaps $B_1, B_2, \ldots, B_t$ in main memory that have recorded the flow dst's traffic for t consecutive periods, our problem is to design an algorithm that can use these bitmaps to estimate $n^* = |S_1 \cap S_2 \ldots \cap S_t|$. Here, the persistent spread n^* is the number of distinct elements that persist through the t time periods and appear in all the bitmaps.

Bitwise OR. For this problem, a possible solution is to calculate the union bitmap $B_1 \vee B_2 \ldots \vee B_t$ by bitwise OR, and extract information from it about $|S_1 \cup S_2 \ldots \cup S_t|$ to assist the estimation of persistent spread (please search for inclusion-exclusion principle). However, this solution has poor accuracy when t is large. This is because the estimation accuracy of a bitmap algorithm depends on the fill rate — the proportion of bits in a bitmap that are set to one: The higher the fill rate, the worse the estimation accuracy [11]. Since the fill rate of union bitmap $B_1 \vee B_2 \ldots \vee B_t$ increases as t grows, any algorithm based on the union will experience the accuracy degradation. The accuracy loss as t value grows will prohibit network operators to configure an arbitrarily large t, which is critical for differentiating persistent elements from transient elements.

Bitwise AND. Instead of the union bitmap, our solution is to calculate the intersection bitmap $B^* = B_1 \wedge B_2 \ldots \wedge B_t$ by bitwise AND. As stated in Eq. (1), each flow element picks a bit in B_i pseudo-randomly by hash function H. Hence, a persistent element always sets the same bit in bitmap B_i, irrelevant of the index i of a time period. If a persistent element sets the jth bit in B_1 to one, then in subsequent bitmaps, the jth bit will be set to one. Hence, we probably can estimate the number of persistent elements, by counting the bits that are "1" in all the bitmaps $B_1, B_2, \ldots, B_t$, or equivalently, the number of "1" bits in the intersection bitmap $B^* = B_1 \wedge B_2 \ldots \wedge B_t$.

While using the intersection bitmap B^*, the main difficulty to achieve satisfactory estimation accuracy is the *false positive probability*, which is the probability for a bit to be assigned to "1" in each time period by different transient elements, making the bit look as if it were set by a persistent element. This phenomenon occurs mostly frequently when the bitmaps $B_1, B_2, \ldots, B_t$ are overly dense with just a small proportion of zero bits (especially when the number of periods t is small). We will address this false positive issue in this paper.

IV. Estimator based on Intersection Bitmap

In this section, based on the intersection bitmap B^*, we present an algorithm to estimate the cardinality of persistent elements $S^* = S_1 \cap S_2 \ldots \cap S_t$, for an arbitrary number of time periods t. In a single period, putting the persistent elements aside, other elements $S_i - S^*$ are called *transient elements*, which are generated by the comes and goes of normal users. We will filter the short-term network traffic, and estimate the number of persistent elements $n^* = |S^*|$.

A. Analysis of Real Network Traces

Before proceeding to detailed analysis, we firstly verify our assumption about rough independence of transient elements in different measurement periods. The verification uses the traces of real network traffic from CAIDA [10]. In the trace files, we have identified tens of servers with low persistent spreads and free from malicious attacks. Hence, for such servers, almost all of their traffics can be regarded as transient elements due to the intersection between normal users and servers. In Table II, we have tested the inter-dependency of transient traffics, in two arbitrary measurement periods. The subtable (a) is about an HTTP server 224.243.38.27/80, and the subtable (b) is for an HTTPS server 224.243.38.27/443. In both of them, the length of a measurement period is configured to seven minutes, and there is a spacing that lasts for three minutes between any two adjacent periods, in order to reduce the chance of normal users' activities crossing the border of two neighboring periods.

With six measurement periods, Subtable (a) lists the cardinality of the intersection of two element sets S_i and S_j for two arbitrary periods i and j: When $i = j$, we show the spread $|S_j|$ of the jth period; When $i > j$, we are supposed to show the intersected spread $|S_i \cap S_j|$ of the two perioids, but we calculate the ratio $\frac{|S_i \cap S_j|}{|S_j|}$ instead, in order to give an impression of how small the dependency between two periods. Subtable (a) shows that, for any pair of non-neighboring periods, their intersected spread is less than 1%, and hence they can be approximated as independent. Subtable (b) demonstrates a similar phenomenon, where the intersected spreads of two

TABLE II: Weak dependency of normal traffics in different measurement periods, if large persistent flows are absent.

(a) HTTP Server 224.243.38.27/80

Two-period Intersection:	1	2	3	4	5	6
1	22604					
2	5.0%	24598				
3	0.8%	4.2%	27561			
4	0.4%	0.9%	4.2%	29426		
5	0.5%	0.5%	0.9%	4.5%	29456	
6	0.3%	0.5%	0.4%	0.8%	4.2%	30489

(b) HTTPS Server 224.243.38.27/443

Two-period Intersection:	1	2	3	4	5	6
1	40205					
2	3.2%	47119				
3	2.0%	3.2%	48433			
4	1.8%	1.9%	3.0%	60050		
5	1.6%	1.7%	1.9%	3.4%	64332	
6	1.7%	1.8%	1.8%	3.6%	3.9%	69356

periods are less than 4%. Hence, when contacting HTTPS servers, although network users are more prone to be heavy users that cross multiple periods, the assumption about rough dependency between different time periods is still valid.

We have also analyzed the traffics of tens of other HTTP and HTTPS servers. The evaluation results are similar: A legitimate user's continuous interaction with a website is pretty short in duration. They only check out necessary information from one website and don't linger for long and jump to another website. For the transient traffic they generated, there exists a rough independence between different periods. There are two key points of establishing such an independence. The first is to configure an appropriate length of measurement periods (e.g. larger than seven minutes), in order to let one period contain a normal user's interaction with a website. The second is to add a decent spacing between neighboring periods (e.g., three minutes), to reduce the chance of a user's activity crossing the borders of two periods. What we have accomplished is to confine normal uses' short-term behaviors within one period. We mainly care about the long-term stealthy activities that span multiple periods in order to degrade a website's performance.

B. Persistent Spread Estimator

In this subsection, we present the formulas of our persistent spread estimator. The inputs are a sequence of bitmaps B_1, B_2, ..., B_t, and their intersection bitmap B^*. There are two cases for a bit in B^* to be set to "1":

1) it contains at least one persistent elements, or
2) it contains none of the persistent elements, but in each time period, it contains at least one transient elements.

The probability of the first case is $1-P^*$, where P^* is the probability for the bit in B^* to contain no persistent elements.

$$P^* = (1-\frac{1}{m})^{n^*} \approx e^{-\frac{n^*}{m}} \tag{2}$$

Here, we have applied the approximation $(1-\frac{1}{m})^n \approx e^{-\frac{n}{m}}$ that works for large m value.

Let P_i be the probability for a bit of B_i to contain no transient elements in the ith period. We have

$$P_i = (1-\frac{1}{m})^{n_i-n^*} \approx e^{-\frac{n_i-n^*}{m}} \quad (1 \le i \le t), \tag{3}$$

where $n_i - n^*$ is the number of transient elements in the ith period. Hence, the probability of the second case is $P^* \prod_{1\le i\le t}(1-P_i)$, or called the false positive probability. Note that our modeling of the false positive probability assumes the rough independence of transient elements at different periods.

Let X_j^* is the event that the jth bit in B^* is set to "1". The probability of X_j^* is

$$Pr\{X_j^*\} = (1-P^*) + P^* \prod\nolimits_{1\le i\le t}(1-P_i).$$

Let Z^* be the proportion of bits in B^* that remain zeros. We have $1-Z^*$ equals the arithmetic mean of m random variables:

$$1 - Z^* \quad = \quad \frac{1}{m} \cdot \sum\nolimits_{j=0}^{m-1} 1_{X_j^*}, \tag{4}$$

where $1_{X_j^*}$ is the indicator function of X_j^*, which equals one when the event X_j^* happens. Since the bits in B^* are mutually independent, $E(1-Z^*) = \frac{1}{m}\sum_{j=0}^{m-1} E(1_{X_j^*}) = E(1_{X_j^*})$. This implies that the expected proportion of bits in B^* that are ones $E(1-Z^*)$ is equal to the probability $Pr\{X_j^*\}$. Hence,

$$E(1-Z^*) \approx (1-P^*) + P^* \prod\nolimits_{1\le i\le t}(1-P_i). \tag{5}$$

By multiplying both sides of Eq. (5) by $(P^*)^{t-1}$, we have

$$(P^*)^{t-1}E(Z^*) \approx (P^*)^t - \prod\nolimits_{1\le i\le t}(P^* - P^*P_i).$$

Combining (2) and (3), we have the following approximation.

$$P_i \approx e^{-\frac{n_i-n^*}{m}} = e^{-\frac{n_i}{m}} \,/\, e^{-\frac{n^*}{m}} \approx E(Z_i) \,/\, P^*$$

Applying the approximation, we have

$$(P^*)^{t-1}E(Z^*) \approx (P^*)^t - \prod\nolimits_{1\le i\le t}\big(P^* - E(Z_i)\big). \tag{6}$$

Since $1-Z^*$ is the arithmetic mean of m independent random variables as shown in Eq. (4), according to the central limit theorem, Z^* approximates a Gaussian distribution. Its variance is inversely proportional to the bitmap size m, which have been proved in Appendix A. Hence, when m is sufficiently large (e.g., a few thousands), we can substitute the mean value $E(Z^*)$ in (6) by an instance value Z^* without producing significant estimation error. By a similar reason, we can replace $E(Z_i)$ by an instance value Z_i. Therefore, we have

$$(\hat{P^*})^{t-1}Z^* \approx (\hat{P^*})^t - \prod\nolimits_{1\le i\le t}(\hat{P^*} - Z_i). \tag{7}$$

Here, an estimation of P^* is denoted by $\hat{P^*}$ with an upper hat.

By observing bitmaps B^* and B_i, we can know Z^* and Z_i, respectively. Hence, there is only one unknown variable $\hat{P^*}$ in Eq. (7). We can solve this equation for $\hat{P^*}$, and then use the relation $P^* \approx e^{-\frac{n^*}{m}}$ in (2) to obtain an estimation $\hat{n^*}$. In the following, we present the formula of the estimation $\hat{n^*}$ for different number of periods t.

- When $t = 1$, equation (7) can be simplified as $Z^* = Z_1$. This is natural because we have $B^* = B_1$ when there is a single period. Since $t = 1$, all the flow elements in bitmap B^* are persistent elements. We can estimate their number from the proportion of bits in B^* that are zeros. Hence,

$$\hat{n^*} = -m \ln(Z^*) \ . \tag{8}$$

- When $t = 2$, equation (7) becomes

$$\begin{aligned} 0 &\approx (\hat{P^*})^2 - (\hat{P^*} - Z_1)(\hat{P^*} - Z_2) - \hat{P^*} Z^* \\ &\approx (Z_1 + Z_2 - Z^*)\hat{P^*} - Z_1 Z_2. \end{aligned}$$

Hence, we have $\hat{P^*} \approx \frac{Z_1 Z_2}{Z_1 + Z_2 - Z^*}$. Combining it with $P^* \approx e^{-\frac{n^*}{m}}$, we can estimate the persistent spread as

$$\begin{aligned} \hat{n^*} &= -m \ln(\hat{P^*}) \approx -m \ln\Big(\frac{Z_1 Z_2}{Z_1 + Z_2 - Z^*}\Big) \\ &= m \ln(Z_1 + Z_2 - Z^*) - m \ln(Z_1) - m \ln(Z_2). \end{aligned} \tag{9}$$

- When $t = 3$, equation (7) can be converted to

$$0 \approx \Big(\sum_{1 \le i \le 3} Z_i - Z^*\Big)(\hat{P^*})^2 - \Big(\sum_{1 \le i < j \le 3} Z_i Z_j\Big)\hat{P^*} + \prod_{1 \le i \le 3} Z_i.$$

We firstly solve the above equation for $\hat{P^*}$, and then use the relation $P^* \approx e^{-\frac{n^*}{m}}$ to estimate persistent spread n^* as

$$\hat{n^*} = m \ln\left(\frac{B - \sqrt{B^2 - 4A\Big(\sum\limits_{1 \le i \le 3} Z_i - Z^*\Big)}}{2A}\right), \tag{10}$$

where

$$A = \prod\nolimits_{1 \le i \le 3} Z_i, \qquad B \ = \ \sum\nolimits_{1 \le i < j \le 3} Z_i Z_j.$$

- When $t \ge 4$, because the order of Eq. (7) about $\hat{P^*}$ grows to at least three, it is complicated to obtain a closed-form estimator. Hence, we propose to solve Eq. (7) by numerical root-finding algorithms, e.g., Newton-Raphson method.
Firstly, we generate an initial guess of $\hat{P^*}$, using $\hat{P^*} \approx Z^*$. This approximation is obtained by dropping the false positive probability $P^* \prod_{0 \le i \le t}(1 - P_i)$ in Eq. (5).
Secondly, we optimize the current value of $\hat{P^*}$, by invoking the following equation iteratively:

$$\hat{P^*} = \hat{P^*} - \frac{z(\hat{P^*})}{z'(\hat{P^*})},$$

where

$$\begin{aligned} z(\hat{P^*}) &= (\hat{P^*})^t - (\hat{P^*})^{t-1} Z^* - \prod\nolimits_{1 \le i \le t}(\hat{P^*} - Z_i), \\ z'(\hat{P^*}) &= t\,(\hat{P^*})^{t-1} - (t-1)(\hat{P^*})^{t-2} Z^* - \\ &\qquad \Big(\prod\nolimits_{1 \le i \le t}(\hat{P^*} - Z_i)\Big)\Big(\sum\nolimits_{1 \le j \le t} \frac{1}{(\hat{P^*} - Z_j)}\Big). \end{aligned}$$

Thirdly, when the optimization process converges, we use the best $\hat{P^*}$ to derive an estimation of the persistent spread.

$$\hat{n^*} = -m \ln \hat{P^*} \tag{11}$$

In summary, for an arbitrary t value, we have presented an equation to calculate the estimation of persistent spread $\hat{n^*}$. In order to shield the difference in the estimation equations of n^*, we define a unified function f_t in the following theorem.

Definition 1 (Bitmap-based Persistent Spread Estimator): Given an arbitrary number of periods t ($t \ge 1$), a unified estimator function to estimate the persistent spread is

$$\hat{n^*} = f_t\big(m, Z^*, \{Z_i\}\big), \tag{12}$$

where Z^* is the proportion of bits of the intersection bitmap B^* that are zeros, Z_i is the zero ratio of bitmap B_i in the ith period ($1 \le i \le t$), and m is the size of each of the bitmaps. When t is 1, 2, 3 or at least 4 respectively, the function f_t corresponds to the Equations (8) (9) (10) or (11).

V. Analysis of Bitmap-based Estimator

In this section, we analyze the bias and variance of our intersection bitmap-based estimation $\hat{n^*}$ in Definition 1.

Firstly, we prove that the estimation $\hat{n^*}$ is asymptotically unbiased when the bitmap size m is sufficiently large. We know that the zero ratio Z^* of bitmap B^* approximates a Gaussian distribution, because Z^* is the arithmetic mean of a large quantity of independent random variables as in Eq. (4). For a similar reason, the zero ratio Z_i of bitmap B_i approximates a Gaussian distribution. From (7), we know that $\hat{P^*}$ is a polynomial function of Z^* and Z_i, with continuous first partial derivatives. According to multivariate delta-method [12], when the bitmap size m is enough large, $\hat{P^*}$ approximately follows a Gaussian distribution, and its expected value $E(\hat{P^*})$ satisfies

$$\big(E(\hat{P^*})\big)^{t-1} E(Z^*) \approx \big(E(\hat{P^*})\big)^t - \prod\nolimits_{1 \le i \le t}\big(E(\hat{P^*}) - E(Z_i)\big),$$

which is obtained by substituting Z^* by $E(Z^*)$, and Z_i by $E(Z_i)$ in Eq. (7). Combining the above formula with Eq. (6), we can derive that $E(\hat{P^*}) \approx P^*$. Therefore, $\hat{P^*}$ approximates a Gaussian distribution whose expected value is P^*. Further, we have $\hat{n^*}$ is a function of $\hat{P^*}$ as $\hat{n^*} = -m \ln(\hat{P^*})$. From delta-method [12], $\hat{n^*}$ approximates a Gaussian distribution with

$$E(\hat{n^*}) \approx -m \ln(E(\hat{P^*})) \approx -m \ln(P^*) = -m \ln(e^{-\frac{n^*}{m}}) = n^*.$$

Therefore, the persistent spread estimation $\hat{n^*}$ is asymptotically unbiased, when the bitmap size m is sufficiently large.

Secondly, we analyze the variance of estimation $\hat{n^*}$ in the following theorem using the tool of Cramér-Rao bound.

Theorem 1 (Variance of Multi-period Estimators): For our persistent spread estimator in Definition 1, its variance is

$$Var(\hat{n^*}) \approx \frac{n^*}{\rho^*} \cdot \frac{1}{(1 - \tilde{P})^2\Big(\frac{1}{\tilde{P}} + \frac{1}{1 - \tilde{P}}\Big)} \ , \tag{13}$$

where $\rho^* = \frac{n^*}{m}$ is the density of persistent elements, $\mathrm{SNR}_i = \frac{n^*}{n_i - n^*}$ is the signal-to-noise ratio in the ith period, and

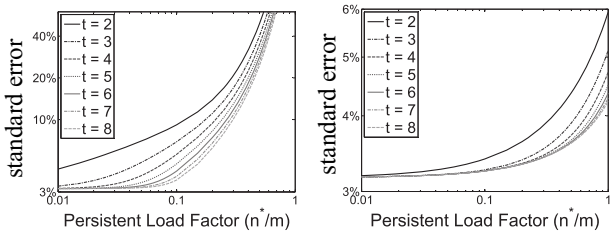
$$\tilde{P} = \big(1 - e^{-\rho^*}\big) + e^{-\rho^*} \prod\nolimits_{1 \le i \le t}\big(1 - e^{-\rho^* \frac{1}{\mathrm{SNR}_i}}\big).$$

Proof: Please check Appendix B for a proof. ■

From Theorem 1, we derive the standard estimation error as

$$\frac{\sqrt{Var(\hat{n^*})}}{n^*} = \frac{1}{\sqrt{m}} \Big/ \sqrt{(\rho^*)^2 \cdot (1 - \tilde{P})^2\Big(\frac{1}{\tilde{P}} + \frac{1}{1 - \tilde{P}}\Big)}.$$

The relative standard error is affected by four factors: (1) density of persistent elements ρ^*, or call it persistent load factor, (2) the number of bits m, (3) signal-to-noise ratio SNR_i, and (4) the number of time periods t. We analyze their impacts by plotting the relative error against these factors in Fig. 2.



(a) For each ith period, $\text{SNR}_i = 0.1$. (b) For each ith period, $\text{SNR}_i = 1$.

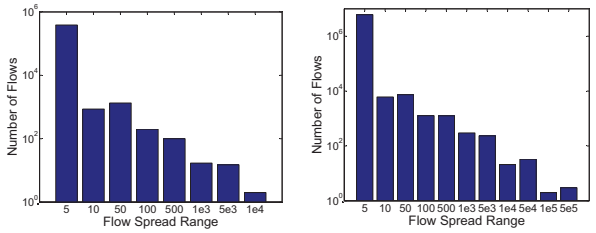
Fig. 2: Accuracy of persistent spread estimation with $n^* = 1000$.

Fig. 2 shows that the estimation accuracy improves as the increase of signal-to-noise level SNR_i, which has been defined as persistent spread n^* divided by the cardinality of transient elements $n_i - n^*$ in the ith period. Subfigure (a) configures the SNR_i as low as 0.1, and the accuracy ranges between 3% and 40+% depending on the load factor. In contrast, subfigure (b) increases SNR_i by ten times to 1. Hence, the accuracy improves and fluctuates between 3% and 6%.

We focus on Fig. 2(b), and it tells us that the estimation accuracy improves as the number of periods t increases. Given more bitmaps $B_1, B_2, \ldots, B_t$, our persistent spread estimator can to reduce the false positive probability $P^* \prod_{1 \le i \le t}(1-P_i)$, and better filter the transient contacts. This plot also shows that the estimation accuracy deteriorates as the density of persistent elements ρ^* increases. The explanation is that our estimation relies on the proportion of zero bits in bitmap B^*. If the density of persistent elements ρ^* grows, B^* will become crowded, and when ρ^* exceeds a bound, the proportion of zero bits in B^* approach zero, which can not be used for accurate estimation.

VI. Multi-virtual Bitmap Estimator

Motivation. In the design of our previous bitmap-based estimator, each flow is allocated with a bitmap to record its elements in a time period, and all the bitmaps are separated and with equal size. Because bitmap algorithm only supports the counting of cardinalities linear to bitmap size [11], this design best fits the case that the flow spreads uniformly distribute. However, the distribution of flow spreads is extremely unbalanced in real networks, especially in core networks. We plot a distribution of flow spreads in Fig. 3, which is obtained from real-world traffic traces from CAIDA [10]. In subfigure (a) where the measurement duration is set to one minute, there are about a million of flows whose spreads are smaller than 100. In contrast, only a few hundreds flows have their spreads larger than 1000. Such an unbalanced distribution of flow spreads can also be witnessed in subfigure (b) where the measurement duration extends to twenty minutes. Throughout the paper, we use the term *mouse flows* to refer to the flows with spreads less than one hundred, and taking the majority of all flows. The term *elephant flows* is used for the flows with extraordinarily larger spreads than the rest. They typically correspond to the server machines with a large number of concurrent users.



(a) Duration = 1 minute. (b) Duration = 20 minutes.

Fig. 3: Flow spread distribution by analyzing CAIDA traces.

Due to the uneven distribution of flow spreads, if allocating all the flows with separated and equal-sized bitmaps, it incurs a significant waste of precious space of on-chip SRAM, which we explain as follows. Since the bitmap method can only count cardinalities linear to bitmap size [11], we have to configure the bitmap size large enough and proportional to the spreads of elephant flows. Otherwise, these bitmaps, when receiving too many elements from elephant flows, have most their bits to be "1", which severely degrades the estimation accuracy. However, we can not predict which flows are elephant flows, and to guarantee the estimation accuracy, we have to allocate all the flows with equal-sized bitmaps that are large enough to accommodate the elephant flows. Therefore, for mouse flows with small spreads, their bitmaps are inevitably sparse with most bits being "0", which causes a significant waste of the expensive SRAM space, especially considering the fact that the majority of flows witnessed by the router are mouse flows.

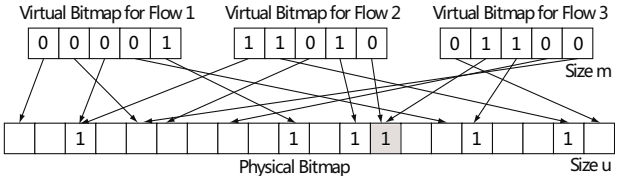


Fig. 4: Multiple virtual bitmaps share bits in a physical bitmap.

To mitigate the memory waste due to uneven distribution of flow spreads, we adopt the idea of *virtualization*: the bitmaps of all the flows are no longer separated but share a common bit pool, which is called the *physical bitmap*. Then, the bitmap of each flow draws its bits pseudo-randomly from the common bit pool, which is called a *virtual bitmap* since it does not physically exist. As illustrated in Fig. 4, the bits of a virtual bitmap uniformly distribute in the physical bitmap. For all the virtual bitmaps, we configure a unified size that is large enough to accommodate elephant flows. Through the bit sharing in physical bitmap, the elephant flows can "borrow" bits from the under-used virtual bitmaps of mouse flows. The physical bitmap is denoted by M, which is an array with u bits allocated from on-chip SRAM. We will describe in detail how to utilize this data structure to estimate the persistent spreads simultaneously for multiple flows.

The idea of virtual bitmaps sharing a physical space has been partially discussed in prior literature [7]. However, they concentrate on estimating the cardinality of a single set. In contrast, we estimate the cardinality of intersection of multiple sets, which are collected in different time domains.

A. Physical Bitmap Encoding

In each time period, the router will observe a large number of traffic flows. For each flow, the router stores its elements into the physical bitmap M, which we explain in details as follows.

Whenever a packet arrives whose header is $\langle src, dst\rangle$, the router uses its destination address to categorize it to the flow dst, and treats its source address src as an element of flow dst, which is mapped to the flow's virtual bitmap. Assume the jth bit in the virtual bitmap has been set to one by the element:

$$j = H(src) \bmod m, \tag{14}$$

where H is a hash function and m is the size of virtual bitmaps which is sufficiently large to accommodate an elephant flow.

According to the bit sharing scheme in Fig. 4, the jth ($0 \le j < m$) bit in the virtual bitmap will be drawn from or mapped to the ith ($0 \le i < u$) bit in the physical bitmap:

$$i = H_{dst}(j) \bmod u,$$

where H_{dst} is a hash function used by the flow dst for bit mapping. It can be implemented from a master hash function H:

$$H_{dst}(j) = H(j \oplus dst), \tag{15}$$

where $\oplus$ is bitwise XOR or string concatenation to combine two key values j and dst. Applying the equation (14), we have

$$H(j \oplus dst) = H\big((H(src) \bmod m) \oplus dst\big)$$

In summary, when the packet $\langle src, dst\rangle$ arrives, the following bit in the physical bitmap M will be set to one:

$$M[i] := 1,$$

where $i = H\big((H(src) \bmod m) \oplus dst\big) \bmod u$.

When the time period terminates, the physical bitmap M will be downloaded from on-chip SRAM to main memory. Assume we have t physical bitmaps, denoted by M_1, M_2, ..., M_t, which correspond to t consecutive time periods.

B. Persistent Spread Estimation

In this subsection, we describe how to use the sequence of physical bitmaps M_1, M_2, ..., M_t, to estimate the persistent spread for a particular flow dst. An intuitive method is that, from an arbitrary physical bitmap M, we can extract a virtual bitmap B that belongs to the flow dst.

$$B = \big\langle M[H_{dst}(0)],\ M[H_{dst}(1)],\ \ldots,\ M[H_{dst}(m-1)]\big\rangle$$

Here, we use the relation that the jth ($0 \le j < m$) bit in virtual bitmap has been mapped to the ith bit in physical bitmap as $i = H_{dst}(j) \bmod u$, and we omit $\bmod u$ for simplicity. Since we have t physical bitmaps M_1, M_2, ..., M_t, we can extract t virtual bitmaps, noted as B_1, B_2, ..., B_t. Then, we can apply our previous algorithm in Section IV-B, to filter the transient elements and estimate the number of persistent elements hiding in the virtual bitmap of flow dst.

However, this method has the problem of overestimating the persistent spread of flow dst. In the flow's virtual bitmap, the persistent elements may not belong to flow dst alone. Since the virtual bitmap of a flow draws bits from a common bit pool that is shared with other flows, the bits in the virtual bitmap may be assigned by other flows to "1". If some of the "1" bits happen to be set by persistent elements coming from other flows, then these bits will be set to "1" in all the virtual bitmaps B_1, B_2, ..., B_t in t time periods, which causes the overestimation of persistent spread of the flow dst.

It may appear that transient elements from other flows may also cause overestimation, since they increase the number of elements n_i that are contained in the virtual bitmap B_i and aggravate the false positive probability. However, when we use zero ratio of B_i to estimate the number of elements in the virtual bitmap of ith period, the estimation result already counts the transient elements coming from other flows. Hence, when we estimate the number of persistent elements in virtual estimator of flow dst, there won't be any overestimation. The major source of overestimating flow dst's persistent spread is the persistent elements from other flows.

Our solution is to compensate the estimation bias due to persistent elements coming from other flows. Let n^* be the number of persistent elements that belong to flow dst, n^*_m be the number of persistent elements in virtual bitmap of flow dst, and n^*_u be the number of persistent elements in physical bitmap M. Our basic idea is that the total number of persistent elements from other flows is $n^*_u - n^*$, and they uniformly distribute in the entire physical bitmap, which are *noises*. Let X be the number of noise elements that are mapped to a bit of physical bitmap M. We know that X follows a binomial distribution: $X \sim Binom(n^*_u - n^*, \frac{1}{u})$. The expected number of noises elements mapped to a virtual bitmap equals to $E(mX)$, where m is the number of bits in a virtual bitmap.

$$E(n^*_m - n^*) = E(mX) = mE(X) = \frac{m}{u}(n^*_u - n^*)$$

According to the laws of large numbers in probability theory, when the number of independent variables m is large enough, the variance $Var(\frac{n^*_m - n^*}{E(n^*_m - n^*)})$ approaches to zero. Hence, when the number of trials m is large, the expected value $E(n^*_m - n^*)$ can be approximated by its instance value $n^*_m - n^*$. Then,

$$n^*_m - n^* \approx E(n^*_m - n^*) = \frac{m}{u}(n^*_u - n^*).$$

By conversion, we have an estimator of persistent spread n^*.

$$n^* \approx \frac{um}{u-m}\left(\frac{n^*_m}{m} - \frac{n^*_u}{u}\right).$$

In summary, our estimator can be divided into three steps.

- First, we estimate n^*_m — the number of persistent elements that are mapped to the virtual bitmap of flow dst:

  $$\hat{n^*_m} = f_t\big(m, Z^*_m, \{Z_{m,i}\}\big)$$

  where f_t is the persistent spread estimator in (12), m is the number of bits in virtual bitmaps, $Z_{m,i}$ is the proportion of bits in the ith virtual bitmap B_i that are zeros, and Z^*_m is the ratio of bits in $B^* = B_1 \wedge B_2 \wedge \ldots \wedge B_t$ that are zeros.

- Second, we estimate n^*_u — the number of persistent elements in physical bitmap:

  $$\hat{n^*_u} = f_t\big(u, Z^*_u, \{Z_{u,i}\}\big),$$

  where f_t is the persistent spread estimator in (12), u is the number of bits in physical bitmap, $Z_{u,i}$ is the proportion

of bits in physical bitmap M_i that are zeros, and Z_i^* is the ratio of bits in $M^* = M_1 \wedge M_2 \wedge \ldots \wedge M_t$ that are zeros.

- Third, we compensate the positive bias in $\hat{n_m^*}$ due to noise persistent elements from other flows, and we obtain the unbiased estimation $\hat{n^*}$ below, for flow dst's persistent spread.

$$\hat{n^*} = \frac{um}{u-m}\left(\frac{\hat{n_m^*}}{m} - \frac{\hat{n_u^*}}{u}\right) \tag{16}$$

VII. Simulation Evaluation

In this section, we use simulation to evaluate the estimators we have proposed: One is based on the intersection of bitmaps, and the other is the multi-virtual bitmaps. The goal of this paper is to design an estimator that is able to use the tight space on on-chip SRAM to deliver high accuracy. Hence, in our experiments, the memory cost, when averaging over all elements appearing in an arrival packet stream, is less than 1 bit per element. The only related work that can work in such tight space is a method based on a continuous variant of Flajolet-Martin sketches, named FMSK for short [5]. We will compare our methods with FMSK in estimation accuracy. We will show the impact of the number of periods t and the signal-to-noise ratio SNR_i on estimation accuracy, which is not quantified by previous works. We will also compare our methods with the aforementioned hash table method storing partial signatures (call it partial hash for short), to show the power of our methods in compressing memory cost.

A. Experiment Setup

We simulate the real-world network traffic using the following parameters. The number of flows that can be observed by the gateway router is configured to 1024, which simulates a small server farm. For a flow, the average number of elements in the ith ($1 \leq i \leq t$) period is configured to 1200, which simulates multiple users concurrently accessing a single server. Some of the flow elements are persistent elements, which exist throughout the t periods, and the rest are transient elements. In each period, we control the ratio of persistent elements to the transient elements by signal-to-noise ratio SNR_i. For these transient elements, we assume that 90% of them stay within one period, and the remaining 10% are heavy users that cross the boundaries between periods.

For fair comparison, we allocate the same size of memory for partial hash, FMSK and our methods. As listed in Table III, each of the three method is given roughly 1.2M bits SRAM, which means each flow gets 1144 bits on average for its spread estimation. FMSK divides these bits into thirty five float numbers, each of which is 32 bits long and can perform the counting independently. Their stochastic averaging is treated as the final estimation. Our bitmap method uses these bits as a bitmap to record the flow elements (whose expected number is 1200). Our multi-virtual bitmap method does not separate the allocated SRAM space into equal-sized bitmaps. It instead lets the virtual bitmaps to share the space. The length of each virtual bitmap is configured as large as 6k bits to accommodate elephant flows. For the most basic method based on partial hashing, we give it 9.1M bits SRAM to show the power of our methods (only having 1.2M bits) in compressing memory cost.

B. Estimation Accuracy and Operating Range

In this subsection, we compare the four methods (listed in Table III) in estimation accuracy and operating range. The comparison results are presented in Figures 5, 6, 7 and 8.

Fig. 5 shows that, although the partial hash method is given 9.1M bits memory that is eight times larger than other methods, its estimation is negatively biased. This is because its operating range is merely $2^4 \times 32 = 512$, where 4 is the size of partial signature stored in one bucket and 32 is the number of buckets allocated for one flow. When the persist spread exceeds this range, it is severely underestimated as depicted in Fig. 5.

Fig. 6 states that FMSK can use only 1.2M bits memory to generate unbiased estimations. However, its accuracy is far from satisfactory. This is because FMSK, similar to Flajolet-Martin sketches [13], have the problem of slow start: it has low inaccuracy when the cardinality to be estimated is on the scale of bits allocated, which is about 1120 bits in the simulation. We will explain later that FMSK has another inadequacy that its accuracy degrades when the number of time periods t grows.

Fig. 7 shows that our bitmap method, when given the same memory of 1.2M bits, can improve estimation accuracy significantly as compared with FMSK. Its shortcoming however is its small operating range: When the persistent spread exceeds a point (about 2000 in Fig. 7), its estimations are strongly biased. This is because, for elephant flows, their bitmaps will receive too much elements, which set most of their bits to one and cause severe bias. This shortcoming can be overcome by our multi-virtual bitmap method, which permits elephant flows to borrow bits from small flows, to extend their operating range. In Fig. 8, this method provides accurate estimations even for persistent spreads as large as 10,000. This is because the size of a virtual bitmap (6k bits) is configured five times larger than the size of a bitmap (1144 bits), as shown in Table III.

C. Impact of Time Period t on Accuracy

An interesting feature of our multi-virtual bitmap method is that its estimation accuracy improves when the number of time periods t increases. Fig. 8 depicts the case of $t = 2$ in the leftmost subfigure, and illustrates $t = 10$ in the rightmost. It is a useful feature that permits network operators to set arbitrarily large t values to differentiate persistent and transient elements.

In contrast, the accuracy of FMSK declines when t value grows, as illustrated Fig. 6. When t grows to 10, its estimation error becomes even larger than 50%. This is because the FMSK method estimates the persistent spreads from the fraction of the

TABLE III: Settings of Algorithm Parameters

Algorithm	Partial Hash	FMSK	Bitmap	Multi-virtual Bitmap
Memory	$\approx$ 9.1 Mbit	$\approx$ 1.2 Mbit	$\approx$ 1.2 Mbit	$\approx$ 1.23 Mbit
Parameters	Flow signature = 8 bit, buckets per flow = 32, source signature = 4 bit.	For a flow, number of buckets = 35. In one bucket, an FMSK = 32 bit.	For one flow, the size of each bitmap = 1144 bit.	virtual bitmap size = 6 kbit, physical bitmap size = 1.23 Mbit.

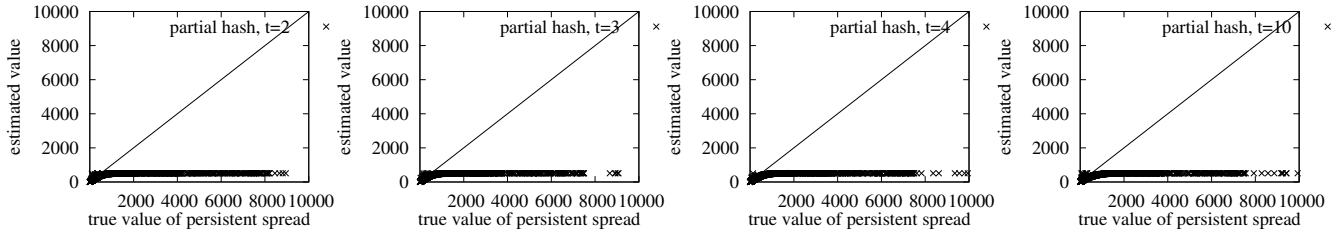


Fig. 5: Persistent spread estimation of partial signature, with $\text{SNR}_i = 1$. From left to right, number of periods $t = 2, 3, 4, 10$.

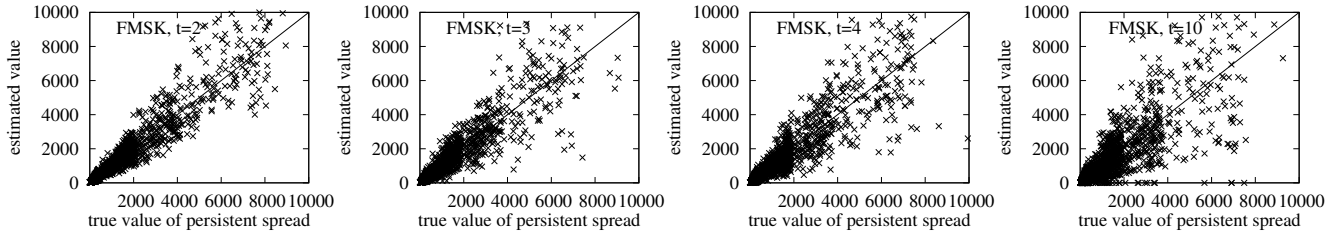


Fig. 6: Persistent spread estimation using FMSK algorithm, with $\text{SNR}_i = 1$. From left to right, number of periods $t = 2, 3, 4, 10$.

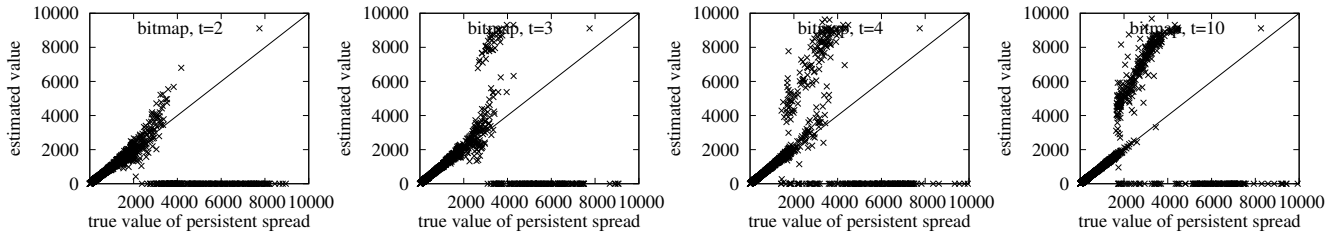


Fig. 7: Persistent spread estimation using bitmap algorithm, with $\text{SNR}_i = 1$. From left to right, number of periods $t = 2, 3, 4, 10$.

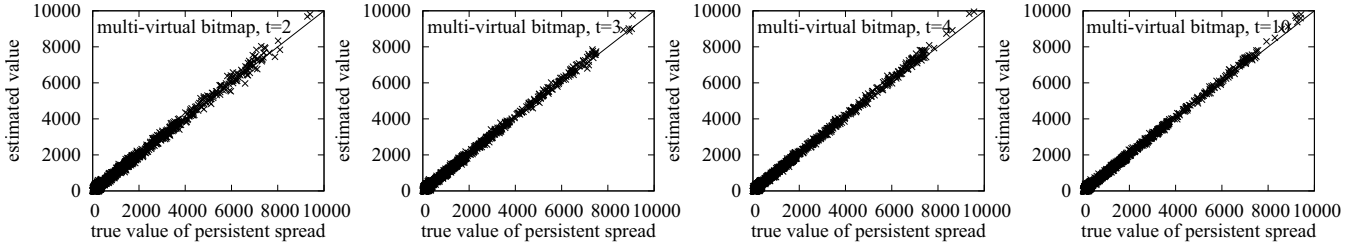


Fig. 8: Persistent spread estimation using multi-virtual bitmap algorithm, with $\text{SNR}_i = 1$.

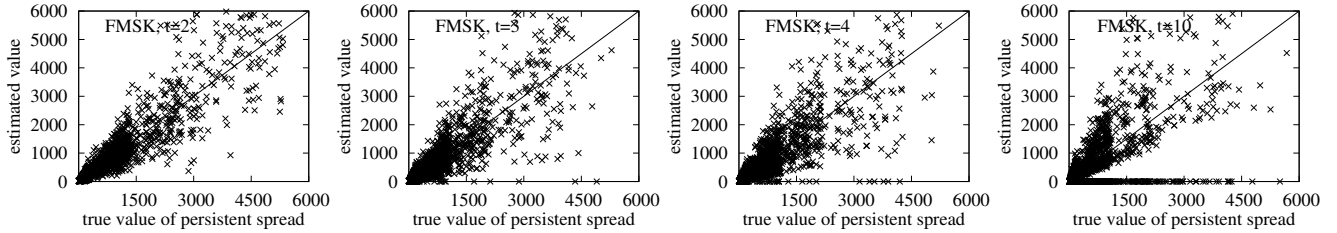


Fig. 9: Persistent spread estimation using FMSK method, with $\text{SNR}_i = 0.4$. From left to right, number of periods $t = 2, 3, 4, 10$.

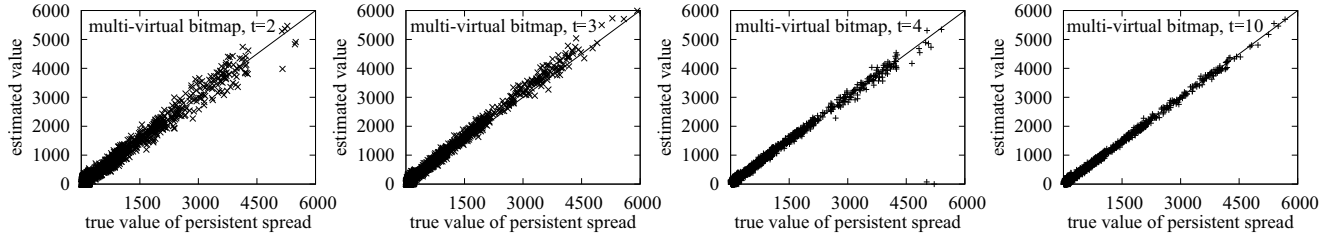


Fig. 10: Persistent spread estimation using multi-virtual bitmap algorithm, with $\text{SNR}_i = 0.4$.

intersected set to the union set of all t periods: $\frac{|S_1 \cap S_2 \ldots \cap S_t|}{|S_1 \cup S_2 \ldots \cup S_t|}$. As t value grows, the size of union set $|S_1 \cup S_2 \ldots \cup S_t|$ expands, which reduces the fraction of intersected set and degrades the estimation accuracy. Our bitmap method is different. It detects the existence of a persistent element from the phenomenon that a bit is set to "1" in all the bit arrays B_1, B_2, ..., B_t. The false positive probability, which the probability that such a bit is occupied only by transient elements, decreases as t value grows, which is the last term $P^* \prod_{1 \le i \le t}(1 - P_i)$ in Eq. (5).

D. Impact of Signal-to-Noise Ratio SNR_i on Accuracy

We present another set of simulation results in Fig. 9 and 10, to study the impact of signal-to-noise ratio on estimation accuracy. The ability of tolerating heavy noise is important, which makes the designed estimator more flexible to use in practice. First, we evaluate the performance of FMSK, by comparing Fig. 6 and 9 which configure the signal-to-noise ratio to 1 and 0.4, respectively. They show that the accuracy of FMSK degrades severely as the noise level increases. Its estimation even becomes biased in the last subfigure of Fig. 9. Second, we evaluate the noise toleration ability of our multi-virtual bitmap method, by comparing Fig. 8 and 10 which configure the signal-to-noise ratio to 1 and 0.4, respectively. The two figures show that the accuracy of our method also degrades, which is consistent with the analysis results in Fig. 2. However, the degree of degradation is pretty modest, and our method can still render satisfactory estimation accuracy when the signal-to-noise ratio is only 0.4 in Fig. 10.

VIII. Related Work

For network traffic measurement, an important branch is passive measurement techniques, which use built-in components of a router or switcher to silently watch the traffic as it passes by. The traversed packets, according to certain fields in the packet header, can be classified to different categories, each of which is called a *flow*. For an individual flow, several kinds of measurements can be taken, including the *flow size* (i.e., the number of packets or bytes or occurrences of certain events in one measurement period) [6], the *flow spread* (i.e., the number of distinct flow elements) [3], [4], [7]. We have proposed to estimate the flow's *persistent spread* (i.e., the number of distinct elements that persist through t time periods), which can be used to detect the long-term stealthy network activities in the background of transient behavior of legitimate users.

Our problem of persistent spread estimation can be applied to detecting stealthy network activities that endure for long periods, e.g., stealthy DDoS attacks, stealthy network scan, and server popularity forging. For this problem, related work exists that detects the stealthy network scan [9]. It however works in spatial domain and detects the presence of a set of hosts that connect to a sufficiently large number of unique destinations within a given time window. In contrast, we detect the network scan from their traffic in temporal domain, and check whether a source node probes different network sections at different time windows. Moreover, our work is a generalized primitive that can detect many other kinds of stealthy activities.

The challenge is the tight constraint on available memory per flow, due to the limited size of on-chip SRAM on line cards and the presence of a large number of flows that share the memory. Many estimators, proposed by previous work for taking per-flow measurement [3], [4], [5], allocate each flow a separated equal-sized data structure. They ignore that flow spreads are extremely imbalanced in real network traffic: Some flows are "elephant flows" whose spreads are thousands of times larger than those of small flows. The counting data structures of elephant flows, due to the injection of too many elements, may become overly dense and have poor estimation. Hence, it is necessary to extend the operating range of counting data structure. There are modern cardinality estimators such as FM sketches [13] and HyperLogLog [14], which appears to be able to handle the elephant flows. However, the accuracy of these estimators depends on the space given. When too many flows exist, each estimator will receive very limited memory on average, which causes accuracy degradation.

Our design is to allow elephant flows borrow memory from small flows, in order to improve their estimation accuracy and extend operating range, To realize the bit sharing among different flows, we construct a virtual bitmap for each flow whose bits uniformly distribute in the overall allocated space (see Fig. 4). Although this idea of virtual bitmap has been discussed by literature [7], [8], their purpose is to estimate the spread of each flow, and only deal with one virtual bitmap for each flow. In contrast, we consider t virtual bitmaps together (collected in t time periods), and estimate their intersection, i.e., the number of elements that persist through the t periods.

For the problem of persistent spread estimation, an important design choice is which data structure should be adopted to record per-flow information. The paper in [5] uses a continuous variant of Flajolet-Martin sketches. We have chosen the well-known bitmap [11], out of two considerations. According to a comparison work in [15], bitmap structure is able to achieve higher accuracy than FM sketches and HyperLogLog, if given enough memory. Moreover, bitmap structure, if enhanced by our multi-virtual bitmaps, can extend the operating range sufficiently large to handle the elephant flows in our application.

IX. Conclusion

In this paper, we have presented a new primitive for passive network traffic measurement, called *persistent spread estimation*, which can help to detect long-term stealthy network activities in the background of short-term activities of legitimate users. To solve this problem in tight memory space, this paper has presented a compact data structure called *multi-virtual bitmaps*, which is suitable to function in the size-limited on-chip SRAM of high-speed routers. The simulation shows that our estimator can use small memory of less than 1 bit per element, to provide satisfactory accuracy and operating range.

When compared with previous work, our estimator brings two key advantages. Its estimation accuracy improves as the number of measurement periods increases, because our method can more effectively filter the short-term behavior of legitimate users. Its operating range of producing effective measurements has been extended if compared with bitmap method. The latter benefit originates from our data structure named multi-virtual bitmaps, which permits elephant flows to "borrow" bits from mouse flows by sharing bits in a common bit pool. (Due to bit sharing among different flows, persistent elements may come from other flows and incur positive estimation bias, and we

have proposed a method to compensate it.) These advantages have been verified by both analysis and experimental results.

ACKNOWLEDGMENT

This work is partially supported by NSF grant CNS-1115548.

REFERENCES

[1] S. Venkatataman, D. Song, P. Gibbons, and A. Blum, "New Streaming Algorithms for Fast Detection of Superspreaders," in *Proc. of NDSS*, Feb. 2005.

[2] Q. Zhao, J. Xu, and A. Kumar, "Detection of Super Sources and Destinations in High-Speed Networks: Algorithms, Analysis and Evaluation," *IEEE JSAC*, vol. 24, October 2006.

[3] C. Estan, G. Varghese, and M. Fish, "Bitmap Algorithms for Counting Active Flows on High-Speed Links," *IEEE/ACM Trans. on Networking (ToN)*, vol. 14, October 2006.

[4] M. Roesch, "Snort–Lightweight Intrusion Detection for Networks," in *Proc. of 13th Systems Administration Conference, USENIX*, 1999.

[5] A. Chen, J. Cao, and T. Bu, "A Simple and Efficient Estimation Method for Stream Expression Cardinalities," in *VLDB*, pp. 171–182, 2007.

[6] Y. Lu, A. Montanari, S. Dharmapurikar, A. Kabbani, and B. Prabhakar, "Counter Braids: A Novel Counter Architecture for Per-flow Measurement," in *Proc. of ACM SIGMETRICS*, 2008.

[7] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a Spread Estimator in Small Memory," *Proc. of IEEE INFOCOM (Review Scores: 5/5/5)*, 2009.

[8] A. Marold, P. Lieven, and B. Scheuermann, "Distributed Probabilistic Network Traffic Measurements," in *17th GI/ITG Conference on Communication in Distributed Systems (KiVS)*, vol. 17, pp. 133–144, 2011.

[9] Y. Gao, Y. Zhao, R. Schweller, S. Venkataraman, Y. Chen, D. Song, and M. Kao, "Detecting Stealthy Spreaders Using Online Outdegree Histograms," in *Proc. of IEEE IWQoS*, pp. 145–153, June 2007.

[10] "The CAIDA UCSD Anonymized 2013 Internet Traces - January 17." http://www.caida.org/data/passive/passive_2013_dataset.xml, 2013.

[11] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A Linear-time Probabilistic Counting Algorithm for Database Applications," *ACM Trans. Database Syst.*, vol. 15, pp. 208–229, June 1990.

[12] G. Casella and R. L. Berger, "Statistical Inference," *2nd Edition, Duxbury Press*, 2002.

[13] P. Flajolet and G. N. Martin, "Probabilistic Counting Algorithms for Database Applications," *J. Comput. Syst. Sci.*, vol. 31, no. 2, 1985.

[14] P. Flajolet, E. Fusy, O. Gandouet, and et al., "Hyperloglog: The Analysis of a Near-optimal Cardinality Estimation Algorithm," in *Proc. of AOFA: The International Conference on Analysis Of Algorithms*, 2007.

[15] A. Metwally, D. Agrawal, and A. E. Abbadi, "Why Go Logarithmic if We Can Go Linear?: Towards Effective Distinct Counting of Search Traffic," in *Proc. of EDBT*, 2008.

APPENDIX A
BIAS ANALYSIS OF BITMAP-BASED SPREAD ESTIMATOR

We prove that our bitmap-based estimator f_t in Definition 1 is asymptotically unbiased. The estimator f_t is obtained in Section IV by solving the following equation set, which has $t+1$ equations and $t+1$ unknowns (i.e., P^* and P_i).

$$E(Z_i) = P^* P_i \qquad (1 \le i \le t)$$
$$E(Z^*) = P^* - P^* \prod\nolimits_{1 \le t \le t} (1 - P_i)$$

Getting P^* by solving the above equation set can be regarded a variant of maximum likelihood estimation. In the process, the only operations that will produce estimation bias are the replacement of expectations $E(Z_i)$ and $E(Z^*)$ by the observations Z_i and Z^*. We prove that, when the bitmap size m is large enough, the replacement produces negligibly small error, and the estimator f_t thus is asymptotically unbiased.

In bitmap B_i of the ith period, the number of zero bits mZ_i follows binomial distribution, since different bits are mutually independent roughly. This binomial distribution can be approximated as Gaussian distribution, when the array size m is sufficiently large [11]. For this Gaussian distribution, its mean value $E(mZ_i)$ is $me^{-\frac{n_i}{m}}$, and its variance is

$$Var(mZ_i) = m\, e^{-\frac{n_i}{m}} (1 - (1 + \tfrac{n_i}{m})\, e^{-\frac{n_i}{m}}).$$

Then, we know that the variance $Var(Z_i)$ approaches zero when m is sufficiently large. For similar reasons that the zero ratio Z^* is some kind of stochastic averaging in bitmap B^*, the variance $Var(Z^*)$ approaches zero asymptotically, whose proof is omitted in this paper due to limitations of space.

APPENDIX B
VARIANCE OF BITMAP-BASED SPREAD ESTIMATOR

We prove the estimator variance in Theorem 1. The likelihood function of persistent spread n^* using observation Y^* is

$$\mathcal{L}(n^* \mid Y^*) = (1 - \tilde{P})^{Y^*} \cdot \tilde{P}^{m - Y^*}, \tag{17}$$

where $Y^* = mZ^*$ is the number of zero bits in bitmap B^*, and

$$\begin{aligned} \tilde{P} &= (1 - P^*) + P^* \textstyle\prod_{1 \le i \le t} (1 - P_i) \\ &= \left(1 - e^{-\frac{n^*}{m}}\right) + e^{-\frac{n^*}{m}} \textstyle\prod_{1 \le i \le t} \left(1 - e^{-\frac{n_i - n^*}{m}}\right). \end{aligned} \tag{18}$$

The meaning of (17) is the probability of observing Y^* zero-state bits and $m - Y^*$ one-state bits in the intersection bitmap B^*, given the facts of persisting spread n^* and the signal-to-noise ratios SNR_i in each period. The symbol $\tilde{P}$ denotes the probability for a bit to be one in bitmap B^*.

For any estimator of n^* based on the observation Y^* and $m - Y^*$, its variance satisfies the Cramér-Rao inequality below:

$$Var(\hat{n^*} \mid Y^*) \ge \tfrac{1}{I(n^*)},$$

where $I(n^*)$ is the Fisher information which can be calculated using the likelihood function $\mathcal{L}$ in Eq. (17).

$$I(n^*) = -E\Big[\tfrac{\partial^2 \ln \mathcal{L}(n^* \mid Y^*)}{(\partial n^*)^2}\Big]$$

For the log-likelihood function $\ln \mathcal{L}$ in Eq. (17), its first-order derivative and second-order derivative are as follows.

$$\tfrac{\partial \ln \mathcal{L}}{\partial n^*} = \tfrac{1}{\mathcal{L}} \tfrac{\partial \mathcal{L}}{\partial n^*} \;=\; \tfrac{\partial \tilde{P}}{\partial n^*} \Big(\tfrac{m - Y^*}{\tilde{P}} - \tfrac{Y^*}{1 - \tilde{P}} \Big)$$

$$\tfrac{\partial^2 \ln \mathcal{L}}{(\partial n^*)^2} = \tfrac{\partial^2 \tilde{P}}{(\partial n^*)^2} \Big(\tfrac{m - Y^*}{\tilde{P}} - \tfrac{Y^*}{1 - \tilde{P}} \Big) - \Big(\tfrac{\partial \tilde{P}}{\partial n^*} \Big)^2 \Big(\tfrac{m - Y^*}{\tilde{P}^2} + \tfrac{Y^*}{(1 - \tilde{P})^2} \Big)$$

Because the expected value of Y^* is $E(Y^*) = m(1 - \tilde{P})$, the expected value of $\frac{m - Y^*}{\tilde{P}} - \frac{Y^*}{1 - \tilde{P}}$ equals zero. Hence,

$$I(n^*) \;=\; -E\Big[\tfrac{\partial^2 \ln \mathcal{L}(n^* \mid Y^*)}{(\partial n^*)^2}\Big] = \Big(\tfrac{\partial \tilde{P}}{\partial n^*} \Big)^2 \Big(\tfrac{m}{\tilde{P}} + \tfrac{m}{1 - \tilde{P}} \Big). \tag{19}$$

The first-order derivative $\frac{\partial \tilde{P}}{\partial n^*}$ required by the above equation can be derived from Eq. (18), assuming that the signal n^* is independent with the noise $n_i - n^*$ (i.e., $\frac{\partial (n_i - n^*)}{\partial n^*} = 0$).

$$\tfrac{\partial \tilde{P}}{\partial n^*} = \tfrac{1}{m} \Big[e^{-\frac{n^*}{m}} - e^{-\frac{n^*}{m}} \prod_{1 \le i \le t} \left(1 - e^{-\frac{n^*}{m} \frac{1}{\text{SNR}_i}}\right) \Big] = \tfrac{1}{m} (1 - \tilde{P})$$

Finally, by replacing $\frac{\partial \tilde{P}}{\partial n^*}$ in (19) with $\frac{1}{m}(1 - \tilde{P})$ and then using the relation $Var(\hat{n^*} \mid Y^*) \ge \frac{1}{I(n^*)}$, we can obtain the inequality for estimator variance in Theorem 1.