SLEUTH: Single-pubLisher attack dEtection Using correlaTion Hunting *

Ahmed Metwally † ‡
metwally@cs.ucsb.edu

Fatih Emekçi §
femekci@linkedin.com

Divyakant Agrawal ‡ Amr El Abbadi ‡
{agrawal, amr}@cs.ucsb.edu

‡ Department of Computer Science
University of California at Santa Barbara
Santa Barbara CA 93106

§ LinkedIn LTD
2029 Stierlin Court
Mountain View CA 94043

ABSTRACT

Several data management challenges arise in the context of Internet advertising networks, where Internet advertisers pay Internet publishers to display advertisements on their Web sites and drive traffic to the advertisers from surfers' clicks. Although advertisers can target appropriate market segments, the model allows dishonest publishers to defraud the advertisers by simulating fake traffic to their own sites to claim more revenue. This paper addresses the case of publishers launching fraud attacks from numerous machines, which is the most widespread scenario. The difficulty of uncovering these attacks is proportional to the number of machines and resources exploited by the fraudsters. In general, detecting this class of fraud entails solving a new data mining problem, which is finding correlations in multidimensional data. Since the dimensions have large cardinalities, the search space is huge, which has long allowed dishonest publishers to *inflate* their traffic, and deplete the advertisers' advertising budgets. We devise the approximate SLEUTH algorithms to solve the problem efficiently, and uncover single-publisher frauds. We demonstrate the effectiveness of SLEUTH both analytically and by reporting some of its results on the Fastclick network, where numerous fraudsters were discovered.

1. INTRODUCTION

Internet advertising is crucial for the success of the entire Internet, and has evolved into an ideal choice for small and large businesses to target their advertisements to the appropriate customers on the fly. Hence, Internet advertising contributes to the well being of e-commerce. An Internet advertiser, such as eBay, provides an advertising commissioner, say, Valueclick, with its advertisements, allocates a budget, and sets a commission for each customer action, e.g. advertisement clicking, auction bidding, or purchasing. The Internet publishers, for instance neopets.com or myspace.com, motivated by the commission paid by the advertisers, contract with the commissioner to display advertisements on their sites. Advertising covers the running expenses of such sites, and thus, advertising also supports the intellectual value of the Internet.

The main orchestrators in this setting are the commissioners between publishers and advertisers, whose servers are the backstage for targeting and budgeting. Whenever a surfer visits a publisher's site, the surfer is referred to the commissioner's servers. The commissioner picks an advertisement to display on the publisher's site, and logs the impression (advertisement rendering). If the surfer clicks the advertisement, (s)he is referred to the commissioner that logs the click, and *clicks-through* the surfer to the advertiser's site.

Since publishers are paid by the traffic they drive to advertisers, there is an incentive for fraudsters to *inflate* the number of impressions and clicks their sites generate [1, 3, 14, 18, 25, 27, 28, 30, 34]. Even more, on real networks, we notice some fraudsters disguised as publishers, hijack some sites (mirror them under different domains), sign up online with the commissioners for those sites, provide a PayPal® account for anonymity, and start simulating traffic and earning revenue. Equally important is the hazard of dishonest advertisers simulating clicks on the advertisements of their competitors to deplete their advertising budgets [23, 31], which impedes the campaigns of their competitors. Failing to identify fraudulent traffic results in bad reputation for the commissioner, and sometimes in paying forfeitures to advertisers [19, 20]. This kind of fraud is jeopardizing, not only the industry of search engines and Internet advertising, but also the entire Internet [13].

Modern online advertising dynamics entail detecting fraud in this huge traffic environment in near real-time, as illustrated in Figure 1. Since advertisers monitor their campaigns online, commissioners provide advertisers with (usually) hourly campaign reports about the traffic and expenditure of campaigns. These campaign reports should be almost exact since advertisers change their budgets (or bids on keywords if applicable) based on the traffic quality and cost [24]. Hence, it is desirable to detect fraud, for the (hourly) campaign report window, in time less than the span of the window. If fraud detection lags behind by several window spans, advertisers experience poor quality of service, since their budgets will be managed based on misleading campaign reports that reflect both fraudulent and legitimate traffic. Moreover, debiting advertisers' accounts, and later crediting them for detected fraudulent payments, incurs high accounting overhead, and makes fraud detection non-transparent to advertisers. From a data mining perspective, the problem scale is challenging, since an average-sized commissioner receives around 70M records hourly. Thus, a fraud detection system should process each traffic entry in no more than 50μs, which allows for only very efficient algorithms.

*This work was supported in part by NSF under grants IIS 02-23022, and CNF 04-23336.

†This work was done while the first author was an intern at Fastclick, Inc., a Valueclick company. The author is now with the AdSpam team at Google Inc.

PVLDB '08, August 23-28, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 978-1-60558-306-8/08/08

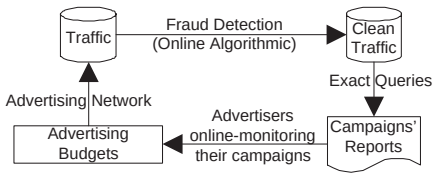

Figure 1: A Framework for How Near Real-Time Fraud Detection Fits into the Advertising Framework

We focus primarily on publishers' fraud where a publisher tries to increase its revenue with fraud traffic. The detection approach and mechanism can be generalized to advertisers' fraud. Our approach for detecting fraud is based on mining the commissioner's traffic logs. The main premise of our traffic analysis approach is to draw correlation between the fraudster's site and the machines used in the attack. To circumvent log analysis, fraudsters strive to dilute the strong correlation between their sites and the machines from which the attacks are launched [29]. This can be done either on the side of the attacking machines or on the side of the attackers' sites. Hence, an attack is either performed by a single-publisher who generates traffic from numerous machines, possibly while obliterating or frequently changing the identification of these machines; or by a group of fraudsters sharing their resources to launch a *coalition* attack. This paper, concentrates on single-publisher attacks, which is the most common case in advertising networks. Detecting single-publisher attacks complements our *coalition* attack discovery algorithms proposed in [28]. Detecting these two classes of attacks leaves no chances for fraudsters to escape. The proposed framework was deployed on one of the major world networks, Fastclick, and was successfully used in fraud detection.

This paper models discovering single-publisher attacks as a new problem of finding correlations in multidimensional data. We devise the SLEUTH algorithms for detecting single-publisher attacks in their most general form. The tunability, effectiveness, and accuracy of the SLEUTH algorithms are demonstrated through comprehensive experiments on the Fastclick network. The SLEUTH algorithms discovered several suspects, most of which were verified manually to be fraudsters, and their contracts were terminated.

We start by describing the economics of online single-publisher attack detection in § 2. § 3 presents the detection approach for single-publisher attacks. The problem model is built in § 4. We devise the *2-Pass-SLEUTH* and *1-Pass-SLEUTH* algorithms in § 5 and § 6, respectively, and contrast their performance in § 7. Our experimental results are reported in § 8. We present the related work in § 9, and we conclude in § 10.

2. ECONOMICS OF SINGLE-PUBLISHER ATTACKS

We start by establishing the association between the number of machines used in a single-publisher attack and the difficulty of detecting the attack. Then, we discuss the economics of detecting single-publisher attacks as an optimization problem.

2.1 The Association Between Attack Sophistication and Detection Difficulty

Understandably, the difficulty of detecting an attack increases as the number of machines from which the attack is launched increases. In its simplest form, launching an attack from one machine, identified by one cookie-ID, can be detected trivially by checking for duplicate impressions and clicks [25]. However, launching an attack from multiple machines is much harder to detect, since the detection algorithm has to examine the relationship between each publisher and all the machines generating its traffic.

Although the straightforward use of network anonymization, e.g. tor.eff.org, is attractive for inexperienced fraudsters, it is not effective. Those services were designed to protect surfers' privacy. Hence, they block surfers' cookies [4]. Therefore, using network anonymization can be trivially detected by monitoring the percentage of cookie-less traffic per publisher and investigating publishers whose traffic deviates from the norm. Similarly, on real networks, we notice some novice fraudsters generating a lot of traffic from ISPs that assign virtual IP addresses to surfers, such as AOL®, in order to hide among legitimate traffic. However, the ranges of IPs of those ISPs are well known, and again, the ratio of the traffic of any publisher received from those ISPs is highly stable across all the publishers. Hence, such attacks are easily detected by examining the ratio of the traffic received from ISPs assigning virtual IPs as compared to the entire publisher's traffic.

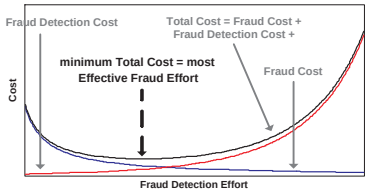

Figure 2: Variation Trends of Total Cost of Fraud and Detection with Fraud Detection Effort

Hard-to-detect attacks from several machines could be costly and unscalable to launch. To have a normal percentage of cookie-less traffic, and a normal percentage of non-virtual ISPs' IPs, fraudsters are motivated to either own the attacking machines, or to control botnets of real surfers' machines through Trojans in order to use IPs and cookies of those real surfers. Even when botnets are used, attacks launched from a botnet of tens of thousands of machines infected by top-notch Trojans are more sporadic than attacks from tens of machines infected by off-the-shelf Trojans for two reasons. First, off-the-shelf Trojan code that is more popular is easily detected with anti-virus software. Hence, the majority of botnets cannot grow to huge sizes. Second, small botnets can still be powerful if coupled with high bandwidth [7]. Therefore, launching scalable attacks entails high cost or requires sophisticated Trojan-writing skills. This establishes the association between the difficulty of launching an attack and the difficulty of detecting it.

2.2 Minimizing the Damage of Single-Publisher Fraud Detection

This association on the fraudsters' side has its ramifications on the side of the detection approach, since most attacks are not highly sophisticated. On the other hand, as we try to detect more sophisticated attacks, we report more honest publishers as suspects (false positives). Discarding all suspects discounts massive traffic from honest suspects and forgoes the corresponding lawful revenue. Reducing the forgone lawful revenue requires scrutinizing the traffic and sales history of suspects via costly human investigation.

From Figure 1, this is even more aggravated by the risk of not producing accurate campaign reports to advertisers in a timely fashion if not enough investigation workforce is employed [13, 24]. Hence, detecting more sophisticated attacks with very few honest suspects and producing accurate timely campaign reports are two conflicting goals. This dilemma incurs tremendous time-critical investigation cost on the commissioners, which raises the advertising cost. In the cases where the prices of the campaigns are fixed by the commissioner and not determined through auctions, it might force advertisers to switch to other commissioners that have less advertising cost. On the other hand, not detecting any attacks at all reduces the campaigns' effectiveness, and hence, raises the advertising cost of the commissioner as compared to its competitors. This also drives advertisers away from the commissioner.

Therefore, a commissioner typically has two extremes on the detection spectrum as shown in Figure 2. One extreme is to employ a

very aggressive detection algorithm that reports most publishers as suspects, and employ a massive number of human investigators in order to detect attacks from any number of machines, and produce timely campaign reports to advertisers. The other extreme is to employ a very lenient detection algorithm that reports few suspects, and deal with the cost of fraud attacks that jeopardizes the commissioner's effectiveness and reputation. Typically, the commissioner aims at reducing its suspects, to save on the human cost, while detecting the majority of attacks, which are not highly sophisticated. This corresponds to the trough of the total cost curve in Figure 2.

2.3 Research Objective

While exactly calculating the optimal total fraud cost is out of the scope of this work because of lack of measured and estimated costs of fraud and detection, we provide a tool for the commissioner to find the *sweet spot* where the total cost curve becomes almost minimal. The goal of this work is to provide an algorithmic tradeoff between detecting attacks from a larger number of machines versus decreasing the number of honest suspects reported as fraudulent.

Through this research, we reduce the total fraud cost by devising an effective detection mechanism whose aggressiveness can be tuned to explore the tradeoff between the fraud and detection costs, in order to minimize the total cost of single-publisher attacks.

3. DETECTION APPROACH FOR SINGLE-PUBLISHER ATTACKS

We now discuss the first steps towards detecting single-publisher attacks. We develop an approach based on how fraudsters can use their resources to launch scalable attacks. We then use this development to present our tunable detection methodology.

3.1 Efficiently Using Resources

Although the fraudster's resources can be limited, there are intelligent ways to increase the sophistication of the attack. Attacks can be made very difficult to detect by frequently changing the identification (IP or cookie ID) of the attacking machines so that the traffic appears as if coming from a large number of machines. As fraud detectives, this makes associating the attacking machines with the fraudster very challenging.

3.1.1 Frequently Changing the IP

Although the attack might be generated from a limited number of machines, say 10, the attacker can change the real IP addresses assigned by the ISP to the attacking machines. Every time a connection (e.g. dial-up) is made to the ISP, the ISP assigns a single real IP address to the account. There is a chance that disconnecting and reconnecting changes the IP address assigned to the machine. However, most ISPs start a session with every connection and assign a new IP only after session expiration. The duration of such sessions is around 30 minutes [17]. Therefore, attackers have to disconnect, wait for the session to expire, and then reconnect again to be assigned a new real IP. This activity throttles the attacking machine. Even if the fraudsters control a moderate number of machines, say 100, their bandwidth will be throttled.

As mentioned earlier, connecting to the Internet through an ISP that uses virtual IPs is trivially detected. Using real IPs and changing them slows down the attack. This makes changing the cookie IDs of the machines the most popular way to scale up the attack.

3.1.2 Frequently Changing the Cookie: The NAT-Masquerading Attack

To make the best use of the IPs, the fraudster can frequently change the cookie identification of the attacking machines, and hence, make the traffic resemble that of Network Address Translation (NAT) boxes. We demonstrate one such attack in Figure 3.

The *NAT-Masquerading* Attack
begin
  For each machine connected to the Internet through a distinct IP
    Phase 0: Connecting to the Internet Anonymously
      Connect to the Internet through a proxy server
    Phase 1: Collecting Cookies
      Repeat several times
        Load publisher's site
        Move assigned cookie to *Bank* Folder
    Phase 2: Faking Traffic
      Repeat forever
        Move a random cookie from *Bank* Folder to *Cookies* Folder
        Simulate an impression or a click
end;

Figure 3: The *NAT-Masquerading* Attack

The *NAT-Masquerading* attack is one way the attackers can increase the number of cookies used in the attack. For each machine connected to the Internet through a unique IP, the attacker starts without a cookie, requests the publisher's page, and thus the machine is assigned a valid cookie with a new ID from the commissioner. This cookie is automatically stored in the "Cookies" folder. The script moves this cookie to a different folder, and repeats the process many times. After the machine ends up with a bank of valid cookies that were assigned by the commissioner, the script selects a random cookie from the cookie bank, returns it to the "Cookies" folder, and makes a fake impression and/or click with this cookie, whose ID is reported to the commissioner. The script returns the cookie to the bank folder, selects another cookie, and so on.

3.1.3 The NAT-Masquerading Attack Strengths

Attacks like *NAT-Masquerading* are very attractive for fraudsters. A fraudster only needs a small number of IPs, and a script that is easy to write. The traffic can pass as normal cookie-identified traffic from numerous surfers behind some NAT boxes. The attack scales well even from a small pool of IPs.

In addition to its simplicity, and inexpensiveness, we discovered that the *NAT-Masquerading* attack is widely used since it is difficult for commissioners to detect, due to the following reasons:

1. **Duplicate detection of cookies is not effective.** The *NAT-Masquerading* attack interleaves the cookie actions to widely separate the activities of the same cookie. Hence, the attack is denser, while impressions and clicks are not duplicated in a small window of time [25].
2. **Duplicate detection of IPs is not appropriate.** Because this attack discredits cookies information, we can only utilize IPs-based signals to detect the attack. In contrast to cookies, IPs could be shared by several machines. Therefore, duplicate detection of IPs would erroneously discard legitimate NAT traffic, and reduce the commissioner's revenue.
3. **Difficulty in matching IPs with fraudsters.** The traffic of a fraudster could be a mixture of legitimate traffic and fraudulent traffic coming from various IPs. In addition, if the fraudster launches his attack through proxy servers, the traffic from the IPs of the proxy server could be a mixture of the fraudster's traffic and genuine traffic of other publishers.

NAT-Masquerading is the most generalized single-publisher attack using any set of IP addresses. We state the following proposition without proof.

PROPOSITION 1. *Using the same number of IPs, any attack using zero or one cookie, can also be detected using the same mechanism that detects attacks with many cookies.*

3.2 Shifting the Focus from Cookies to IPs

To detect the generalized single-publisher attacks, the detection algorithm should go beyond examining cookies. As fraud detectives, we should not rely solely on detecting traffic entries with the

same cookies [25]. Since the general forms of attacks, like *NAT-Masquerading*, reduce the reliability of cookies information, we should examine the relationship between publishers and IPs. We should identify outlier activities of publishers with respect to IPs[1]. Such abnormalities reflect all attacks as discussed next.

One such abnormality could be a publisher whose traffic is monopolized by a few IPs. This could be a fraudster using these IP to launch an attack. However, this measure is overly aggressive, since some IPs, like those of legitimate NAT boxes, are shared by numerous computers, and could generate most of a publisher's traffic.

The complementary abnormal behavior, of some IPs directing a considerable portion of their traffic to a specific site, could reveal IPs used by a fraudster. Yet, it could also be a site, like myspace.com[2], which is highly popular among numerous IPs.

Thus, we need to normalize the traffic and detect active pairs of publishers and IPs. We are searching for correlations that do not reflect the average activity of the IP and the publisher. An inactive IP should be generating little traffic for most publishers. Similarly, an unpopular site should receive slim traffic from most IPs. We are looking for unpopular sites receiving most of their traffic from inactive IPs that are only active for this specific sites, i.e., sites and IPs that are strongly coupled.

4. PROBLEM FORMALIZATION

Now, we build a model for detecting attacks from several IPs.

4.1 Modeling the Problem

Abstractly, the incoming traffic entries can be viewed as independent elements with two dimensions, the publisher, and the IP. The event of generating a traffic entry for a publisher, x, has sample probability $P(x)$. Similarly, the occurrence of an IP, y, has sample probability $P(y)$. To model the problem, we make the following assumption.

ASSUMPTION 1. *The probabilities, $P(x)$ and $P(y)$, of observing x and y, respectively, are normally independent.*

That is $P(x \wedge y)$, the sample probability of publisher x receiving a traffic entry from IP y, is given by $P(x)P(y)$. For example, in a dataset, let x have $P(x) = 0.001$, and y have $P(y) = 2.5{*}10^{-4}$, if x and y are uncorrelated, the commissioner should receive a traffic entry from IP y to publisher x with probability $2.5 * 10^{-7}$.

We model the problem of detecting single-publisher attacks as identifying publishers correlated with several IPs, which clearly violates Assumption 1. As we seek weaker correlations, we report more honest suspects, though such anomalies are the primary suspects that could be investigated, a costly time-critical process. Assumption 1 is validated through real network analysis in § 8.2.1, where we report the number of honest suspects of this assumption.

This is a manifestation of the tradeoff between discovering more single-publisher attacks, and minimizing the cost of fraud detection, as discussed in § 2.2. The more attacks we try to discover, the weaker the correlations we target, the more the honest suspects, and the higher the detection cost.

Based on Assumption 1, there should exist a high correlation between a fraudster and the IPs s(he) exploits. Hence, for each publisher, we should search for IPs that generate a significant portion of this publisher's traffic.

However, this metric might be overly aggressive, since some active IPs naturally generate significant portions of the traffic of numerous publishers. Therefore, we should not suspect IPs which are active for several publishers. In other words, we report a correlated pair of a publisher and an IP only if the publisher receives a significant portion of its traffic from this IP, while the IP is driving a non-trivial percentage of its traffic to the publisher's site.

To model the solution, we borrow a definition from the data streams literature, and build on it. We define a correlated pair as one where the IP is a *frequent*[3] element for the publisher, and the publisher is a *frequent* element for the IP.

4.2 Formal Problem Definition

We formulate the problem of detecting *correlations* in a 2-dimensional stream as follows. Given a stream S of size N of 2-dimensional data entries, and two user[4] defined *frequency thresholds* ϕ, and ψ, both in the interval $(0, 1)$; find all pairs, such that for each discovered pair, (x, y), it is true that $F(x, y) > \lceil \phi F(x) \rceil$, and that $F(x, y) > \lceil \psi F(y) \rceil$; where $F(x, y)$ is the number of occurrences of the pair (x, y) in S; $F(x)$ is the aggregate number of occurrences of all the pairs with x as its first dimension value; and $F(y)$ is the aggregate number of occurrences of all the pairs with y as its second dimension value.

For notational purposes, we denote the stream of pairs which have publisher x as its first dimension, S_x, and call it the traffic of publisher x. Similarly, S_y stands for the traffic of IP y. $F(x)$ is the size of S_x, and $F(y)$ is the size of S_y. We also denote the set of distinct publishers in the first dimension A_1; and the set of distinct IPs in the second dimension A_2. We say an IP y is $\phi-$*frequent* for publisher x, if $F(x, y) > \lceil \phi F(x) \rceil$; and publisher x is $\psi-$*frequent* for IP y, if $F(x, y) > \lceil \psi F(y) \rceil$.

The two frequency thresholds ϕ, and ψ are the means by which the commissioner adjusts the aggressiveness of the algorithm. Notice that the thresholds and the algorithms presented hereafter can be tuned according to the nature of publishers and IPs. That is, the commissioner can group the publishers based on their traffic size or the level of confidence, and sets different ϕ thresholds for different groups of publishers. Similarly different ψ thresholds can be used for different groups of IPs. However, for simplicity, we assume one ϕ across all publishers and one ψ across all IPs.

The higher the ϕ, and ψ, the stronger the correlation sought, the fewer the honest suspects, and the lower the fraud detection cost as discussed in § 2.2.

4.3 To Stream or Not to Stream

We have borrowed the problem of frequent elements on data streams to formalize the problem of finding correlation in 2-dimensional streams. In reality, some networks might relax the requirement of detecting fraud in a stream environment. While still analyzing traffic in near real-time, the goal could be relaxed to process the traffic window in time that is less than the window span. This relaxation on the application side leaves room for multi-pass algorithms.

To use a single-pass or multi-pass algorithms is a network design decision. We expect multi-pass algorithms to be more accurate than single-pass algorithms, due to their ability of verifying estimates of frequencies on several passes. Higher accuracy means fewer suspects to manually investigate, and lower detection cost.

On the other hand, from an application perspective, a single-pass algorithm allows reporting suspects at any time. In contrast, multi-pass algorithms have to scan the entire stream at least once before reporting suspects when a query is posed, which introduces some latency in query response time. This gives single-pass algorithms an advantage, especially when dealing with large traffic windows, say days or weeks. This property of online responding to queries is

[1]We concentrate on fraud detection, and assume the traffic source IPs are not spoofed. Counteracting spoofing was studied in [5].

[2]Est. 1999, has over 300M accounts, as of February 2008.

[3]The term *frequent* was defined in [22] as an element in a stream of size N, which occurs more than $\lceil \phi N \rceil$ times, where $\phi \in (0, 1)$ is a user predefined *frequency threshold*.

[4]By user, we refer to the commissioner.

crucial for customer satisfaction, if real-time handling of advertisers' complaints is a network design goal. This way, advertisers can contact the commissioner about suspicious traffic, and the commissioner can respond to their concerns almost instantaneously. From a system design standpoint, single-pass algorithms enable cumulatively monitoring of traffic, and issuing queries at any point, without necessarily defining when the analysis window ends *a priori*.

At Fastclick, we decided to employ a single-pass streaming algorithm due to the above reasons. However, we also explored the multi-pass option. We developed, *2-Pass-SLEUTH*, a two-pass algorithm as an effort in this direction. *2-Pass-SLEUTH* was developed further into our streaming *1-Pass-SLEUTH* algorithm. Before we describe the algorithms in § 5 and § 6, we comment on the infeasibility of an exact solution in a stream environment.

4.4 Infeasibility of an Exact Stream Solution

The problem of exactly detecting correlations is infeasible on streams, as shown in Theorem 1.

THEOREM 1. *Any exact solution for the problem of detecting* correlations *in multidimensional streams requires keeping exact and complete information about all pairs in the stream.*

Proof. The proof is by contradiction for 2-dimensional streams, and the generalization for higher dimensions is straightforward. Given a 1-dimensional stream, $q_1, q_2, \ldots, q_I, \ldots, q_N$, of size N, construct a 2-dimensional stream of pairs of size N as follows, $(q, q_1), (q, q_2), \ldots, (q, q_I), \ldots, (q, q_N)$, by appending some element, q, as the first value for all pairs. An answer to the query about correlations in the constructed stream with thresholds ϕ and $\psi = 1 - \frac{1}{N}$ can be directly translated into an answer to a query about frequent elements in the original stream with threshold ϕ. Answering the exact query about the constructed 2-dimensional stream without complete information contradicts the fact that finding exact frequent elements in a stream requires complete information about all elements [6, 11]. □

From Theorem 1, an exact solution entails keeping complete information about the traffic. The traffic can be stored in a large 2-dimensional array of publishers and IPs. Every time the commissioner receives a new impression or click, the corresponding cell gets incremented. Storing an array of the traffic of publishers and IPs is infeasible. There are approximately $2^{32} = 4$G IPs, and an average commissioner has around 50,000 publishers. An array with a traffic counter for each pair will contain more than $214 * 10^{12}$ cells. If each cell is a 4-Byte integer, the array size is approximately 1 PetaByte. The other extreme is trading space for speed. Since this array is very sparse, it can be stored using a sparse matrix representation [32]. However, the sparse matrix representation entails access time linear in the cardinality of at least one dimension. This is not suitable for real-time updates and queries.

We are not aware of any stream mining solution for the proposed problem. This is one generalization of the frequent elements [22] in data streams for higher dimensionality. The problem of frequent elements in streams has been generalized before in other contexts. The notion of frequent elements was generalized for hierarchies in [8], where an element can only be frequent if the sum of frequencies of its infrequent children satisfies the threshold. The notion was then re-generalized for higher dimensionality in [9]. A single-dimensional stream was summarized in [27] using a 2-dimensional structure containing antecedents and consequents of associations between frequent pairs of elements that are not widely separated in the stream. However, none of the algorithms in [8, 9, 27] can be modified to handle the problem in hand. Also, we are not aware of any other data mining algorithm that can be, which necessitates devising a new approximate algorithm.

5. DEVELOPING A TWO-PASS SCHEME

In this section, we investigate how to detect single-publisher fraud instances by efficiently identifying correlations in multidimensional data over large domains. We start by proposing a two-pass scheme, *2-Pass-SLEUTH*, in § 5.1. Then, we give its implementation details in § 5.2. We develop *2-Pass-SLEUTH* into the single-pass scheme, *1-Pass-SLEUTH*, in § 6.

5.1 2-Pass-SLEUTH: the Big Picture

In the first *discovery pass*, a set of IPs, the *frequent group* $= \bigcup_{\forall x \in A_1} \{y | (y \in A_2) \wedge (F(x,y) > \lceil \phi F(x) \rceil)\}$, is formed that contains all the IPs found to be $\phi-$*frequent* for at least one publisher. For each publisher, x, a data structure is constructed that keeps track of which IPs are $\phi-$*frequent* for x. After scanning the traffic, the publishers' data structures are queried for $\phi-$*frequent* IPs. The second *verification pass* is made on the traffic to verify that each IP in the *frequent group* generates a high percentage of its traffic for the suspected publisher(s). For each IP, y in the *frequent group*, a data structure is constructed that reports the $\psi-$*frequent* publishers for y. After completing the second pass, all *correlations* $= \{(x,y) | y \in$ *frequent group* $\wedge F(x,y) > \lceil \phi F(x) \rceil \wedge F(x,y) > \lceil \psi F(y) \rceil\}$ are reported.

Although the formal problem is symmetric on both dimensions, we purposefully selected the publishers' dimension to handle in the first pass because the cardinality of the publishers is typically much smaller than that of the IPs (50,000 versus 4G). Our goal is a feasible and more accurate processing in the first pass to lessen the IPs to be examined in the second pass. Thus, the large cardinality of the IPs is reduced considerably from a processing standpoint.

To implement this scheme, we have to first identify frequent elements in one pass. Several approximate schemes [10, 11, 12, 15, 16, 22, 26] have been proposed to solve this problem, any of which can be employed to accurately report IPs that are frequent to specific publishers, as well as publishers who are frequent to specific IPs. We chose *Space-Saving* [26] for the reasons explained in § 6.

5.2 Implementing 2-Pass-SLEUTH

We now briefly describe how to implement the two-pass *2-Pass-SLEUTH* scheme using *Space-Saving* [26]. We explain how *Space-Saving* can be incorporated in the *discovery pass* to output the *frequent group*, and the discussion applies to the *verification pass*.

During the first *discovery pass*, a lightweight *Stream-Summary*$_x$ structure is kept for each publisher, x. *Stream-Summary*$_x$ is employed by *Space-Saving* to maintain partial information about exactly m IPs of interest, $e_1, e_2, \ldots, e_m$, using m counters, where m is as specified later. At any time, each counter is monitoring a specific IP. Although *Stream-Summary* is a hash-based structure, the IPs monitored in *Stream-Summary*$_x$ are sorted by their estimated frequencies. That is, e_1 (e_m) has the highest (lowest) estimated frequency, $Count(x, e_1)$ ($Count(x, e_m)$). If an IP is not monitored, its estimated frequency is 0.

Stream-Summary$_x$ monitors the $\phi-$*frequent* IPs for x with a guaranteed error rate, ϵ. *Space-Saving* guarantees that monitored IPs can have their hits overestimated by no more than $\epsilon F(x)$, and never underestimated. Regardless of the permutation of S_x, the guaranteed error rate, ϵ, decreases with the increase in the number of counters in *Stream-Summary*$_x$, m, and with the data skew. Even if all IPs contribute equally to S_x, i.e., in the worst case of uniform data, a permissible error rate, ϵ, is enforced with m at most $\lceil \frac{1}{\epsilon} \rceil$. In practice, ϵ is set between $\frac{\phi}{10}$ and $\frac{\phi}{100}$.

For every observed pair (x, y) in the traffic, the algorithm selects the *Stream-Summary* that monitors x, and updates *Stream-*

Algorithm: ***Space-Saving*****(*Stream-Summary* SS_x, size m, element y)**
begin
  If y is monitored in SS_x {
    let $Count(x, e_i)$ be the counter of y
    $Count(x, e_i)$ ++;
  }**else**{
    //The replacement step
    let e_m be the element with least hits, min
    Replace e_m with y;
    Assign $\varepsilon(x, y)$ the value min;
    $Count(x, y)$++;
  }
end;

Figure 4: The *Space-Saving* Algorithm

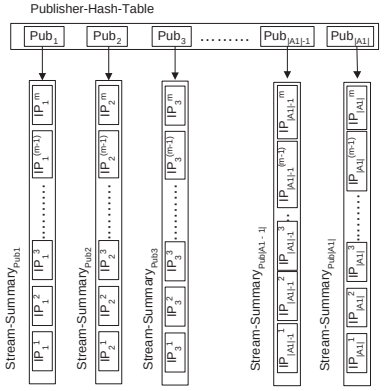

Figure 5: The *Publisher-Hash-Table* Structure

Summary$_x$ using the *Space-Saving* algorithm, as sketched in Figure 4. If there is a counter, $Count(x, e_i)$, in *Stream-Summary*$_x$ assigned to the observed IP, y, i.e., $e_i = y$, then $Count(x, e_i)$ is incremented, and the counter is moved to its correct position in *Stream-Summary*$_x$, in amortized constant time. If y is not monitored, i.e., no counter is assigned to y, then e_m, the IP that currently has the least estimated hits, min, is replaced with y. Since the actual hits of y can be any number between 1 and $min + 1$, *Space-Saving* assigns $Count(x, y)$ the value $min + 1$, since it gives elements the benefit of doubt not to underestimate frequencies. Thus, *Space-Saving* never misses a frequent IP but may report some infrequent IPs as frequent. For each IP, e_i, in *Stream-Summary*$_x$, the algorithm keeps track of its maximum possible over-estimation, $\varepsilon(x, e_i)$, resulting from the initialization of its counter when inserted into the *Stream-Summary*$_x$. That is, when starting to monitor y, set $\varepsilon(x, y)$ to the counter value that was evicted. Thus, at any time and for any IP, e_i, in *Stream-Summary*$_x$, $0 \leq \varepsilon(x, e_i) \leq min \leq \epsilon F(x)$; and $F(x, e_i) \leq Count(x, e_i) \leq (F(x, e_i) + \varepsilon(x, e_i)) \leq F(x, e_i) + \epsilon F(x)$.

As depicted in Figure 5, all publishers' *Stream-Summary* structures are stored in a hash table, *Publisher-Hash-Table*. This ensures constant amortized processing per traffic entry in the first pass.

After the completion of the first pass, for every publisher, x, the IPs in *Stream-Summary*$_x$ are traversed in order of their estimated frequency, and all the IPs are output, until an IP is reached whose frequency is below $\lceil \phi F(x) \rceil$. *Space-Saving* guarantees that all identified $\phi-$*frequent* IPs for any publisher, x, have frequencies no less than $\lceil (\phi - \epsilon) F(x) \rceil$. In addition, an IP y with $F(x, y) > \lceil \phi F(x) \rceil$ is guaranteed to be reported. The *frequent group* is formed as a union of all the reported IPs, and the second *verification pass* is made on the traffic. For each IP, y, in the *frequent group*, the $\psi-$*frequent* publishers for y are found in a way similar to the one used in the *discovery pass*. Concurrent to this process, the frequent IPs discovered for each publisher in the *discovery pass* can be verified to remove uncertainties introduced by the error ϵ. To report anomalies, for each publisher, x, all IPs that are $\phi-$*frequent* for x are checked. For each such IP, y, if x is $\psi-$*frequent* for y, then the pair (x, y) is suspicious.

6. DETECTING CORRELATIONS IN ONE PASS

In § 6.1 and § 6.2, we develop the ideas in § 5 into a one-pass algorithm, *1-Pass-SLEUTH*, and its *Captured-Correlations* structure.

6.1 Merging the Two Passes

The *2-Pass-SLEUTH* scheme presented in § 5 accurately discovers correlations that can be manually inspected to discover fraud. However, it requires two passes on the traffic. In order to make only one pass on the traffic, publishers that are frequent for IPs must be discovered in this single pass. Simultaneous discovery of $\psi-$*frequent* publishers in, S_y, for each IP, y, in the traffic window is not feasible. The reason is, in large samples of real data, the number of distinct IPs occurring in a dataset of length N is roughly found to be $\frac{N}{3}$. That is, collecting information about the traffic of all IPs entails keeping $\frac{N}{3}$ *Stream-Summary* structures; which is an infeasible $O(N)$ approach. Hence, we devote this section to answering the question: *"how to select and monitor IPs that should have their frequent publishers identified in a single pass?"*

6.1.1 Selecting IPs to Monitor

The IPs which should have their traffic monitored are those that would appear in the *frequent group* at the end of the *discovery pass* in the two-pass scheme. Those are the IPs that are $\phi-$*frequent* for at least one publisher. The $\psi-$*frequent* publishers of these IPs should be known in order to verify the suspects discovered on the publishers' dimension. The rest of the IPs, the *infrequent group* $= \{y | (y \in A_2) \wedge (\forall x \in A_1, F(x, y) \leq \lceil \phi F(x) \rceil)\}$, are not $\phi-$*frequent* to any publisher. Due to time and space constraints, it is desirable not to monitor the traffic of IPs in the *infrequent group*, since they have no chance of signaling fraud. In reality, IPs in the *frequent group* can only be known after the *discovery pass*.

We follow a best-effort approach to answer the above question. We integrate the *verification pass* into the *discovery pass* by identifying the IPs of the *frequent group* as early as possible. In Theorem 6, we will establish the correctness of our algorithm by showing that the IPs discovered early to belong to the *frequent group*, are accurate and complete under very reasonable assumptions.

Every IP that becomes $\phi-$*frequent* for at least one publisher in the course of the pass should have its traffic monitored, to verify the suspects discovered on the publishers' dimensions by the end of the pass. The earlier we correctly classify an IP to belong to the *frequent group*, the bigger the analyzed portion of the traffic of this IP. Thus, the answer to the above question lies in the following. *Early and accurate identification of IPs that belong to the* frequent group *is necessary to feasibly and precisely detect correlations*.

However, the set of IPs identified as frequent over time can be unbounded, since some IPs can be identified as $\phi-$*frequent* for some publisher in the course of the pass, but are deemed infrequent by the end of pass. To wisely allocate space for monitoring IPs, the traffic of an IP that becomes $\phi-$*infrequent* for all publishers should stop being monitored.

Hence, we need to employ an accurate frequent elements algorithm that continuously report, for each publisher, which infrequent IPs are becoming frequent, as well as which frequent IPs are becoming infrequent. This is the reason we selected the *Space-Saving* algorithm, since it can incrementally answer continuous queries about frequent elements in a data stream, with negligible constant space and time overheads.

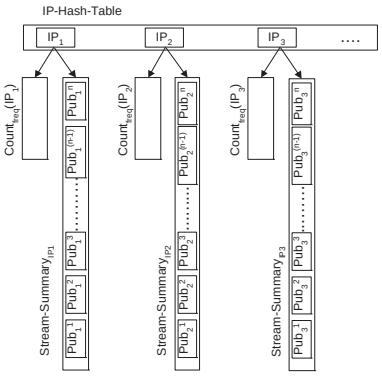

Figure 6: The IP-Hash-Table Structure

6.1.2 Bootstrapping the Monitored IPs

Once we identify an IP, y, as being $\phi-$*frequent* for at least one publisher, the traffic of y should be monitored using a new *Stream-Summary*$_y$ that should be initialized wisely.

The straightforward initialization approach is to collect information about y from all publishers. For each publisher, x, the algorithm should read $Count(x, y)$ from *Stream-Summary*$_x$ and insert it into *Stream-Summary*$_y$. However, this makes the processing time of the traffic entries linear in the number of publishers, which is impractical. We follow a more pragmatic approach.

We sample S_y to populate *Stream-Summary*$_y$. The biggest available unbiased sample of S_y is the part of the traffic not yet observed. Our goal is to make the number of publishers' hits in *Stream-Summary*$_y$ a consistent scale-down of the true hits the publishers have received since the very beginning of the pass. Thus, we create an empty *Stream-Summary*$_y$, and set the counter of x in *Stream-Summary*$_y$, $Count(y, x)$, to 1.

6.2 The 1-Pass-SLEUTH Algorithm and the Captured-Correlations Structure

Now, we describe the *Captured-Correlations* structure, and how it is utilized to summarize correlations between publishers and IPs.

6.2.1 The Captured-Correlations Structure

The *Captured-Correlations* structure consists of two components. First, the *Publisher-Hash-Table* contains a *Stream-Summary* structure for each publisher as explained in § 5.2. *Stream-Summary*$_x$ monitors the $\phi-$*frequent* IPs for x as illustrated in Figure 5.

The second component, as depicted in Figure 6, is the *IP-Hash-Table*, a hash table that monitors the traffic of IPs suspected to belong to the *frequent group*. This table should be searchable by the IP. The number of IPs whose traffic are monitored in *IP-Hash-Table* is limited as shown later in Theorem 5. For each IP, y, in *IP-Hash-Table*, there is a pair, $pair_y$, $\langle Count_{freq}(y)$, *Stream-Summary*$_y\rangle$. The traffic of IP y is monitored with *Stream-Summary*$_y$ that tracks the frequent publishers for y using n counters, where $n = \lceil \frac{1}{\eta} \rceil$, and η is the maximum error rate the commissioner allows in discovering $\psi-$*frequent* publishers in the IPs' dimension.

Meanwhile, for any IP, y, the number of publishers y is currently frequent for is stored in the counter $Count_{freq}(y)$. If an IP, y, is currently frequent for at least one publisher, then it is expected to belong to the *frequent group*, and its traffic should be monitored in *IP-Hash-Table*. The purpose of keeping $Count_{freq}(y)$ is to stop monitoring the traffic of IP y, and free the space used by $pair_y$ once $Count_{freq}(y)$ is decremented to 0. When this happens, y is no more expected to belong to the *frequent group*.

6.2.2 The φ Reduced Threshold

In order to allow for early discovery of the IPs that belong to the *frequent group*, we reduce the frequency at which the publishers' *Stream-Summary* structures report an IP to be frequent or infrequent. For a publisher, x, instead of reporting an IP, y, to be frequent when $Count(x, y)$ reaches $\phi F'(x)$, y is reported as frequent when $Count(x, y)$ reaches $\varphi F'(x)$, where $F'(x)$ is the number of hits seen so far for publisher x, φ is a user specified *reduced threshold*[5], and $\epsilon \leq \varphi \leq \phi$. This enables more accurate monitoring of traffic of IPs in the *frequent group*.

Similarly, the *Stream-Summary* of an IP, y, is deleted from *IP-Hash-Table* only when its frequency drops below the *reduced threshold* for all publishers, i.e., $Count(x, y) < \varphi F'(x)$ for every publisher, x. This delayed deletion of the *Stream-Summary* of an infrequent IP gives it a chance to become frequent again without loosing information about its past traffic.

Using φ instead of ϕ to assign *Stream-Summary* structures to IPs, and incrementing or decrementing their counters in *IP-Hash-Table* entails some overhead, since some IPs in the *infrequent group* will have their traffic monitored. However, as larger portions of the traffic of IPs are monitored, fewer honest suspects are output, which reduces the human cost significantly.

6.2.3 The 1-Pass-SLEUTH Algorithm

The *1-Pass-SLEUTH* algorithm is sketched in Figure 7. For every observed pair (x, y), the algorithm selects the *Stream-Summary* that monitors x, and increments the counter of y in *Stream-Summary*$_x$. The algorithm then checks if there are any IPs in *Stream-Summary*$_x$ that became $\varphi-$*frequent*. Only the last observed IP, y, can become $\varphi-$*frequent*. If this is the case, $Count_{freq}(y)$, is incremented, if y already belongs to *IP-Hash-Table*. Otherwise, y is inserted into *IP-Hash-Table* with $Count_{freq}(y) = 1$ and an empty *Stream-Summary*$_y$ data structure. On the other hand, if there are any IPs in the *Stream-Summary*$_x$ that became $\varphi-$*infrequent*, their counters get decremented in *IP-Hash-Table*. If the counter of any of these IPs reaches 0, the IP gets deleted from *IP-Hash-Table*. Next, *IP-Hash-Table* is searched for IP y. If it exists, publisher x is incremented in its *Stream-Summary*$_y$.

Once the traffic is scanned by *1-Pass-SLEUTH* and *Captured-Correlations* is built, the user can query for correlations. The *Anomaly-Alarm* algorithm that reports correlations efficiently is straightforward. As described in Figure 8, the algorithm checks all the publishers in the first dimension. For each publisher, x, it finds all IPs that are $\phi-$*frequent* for x. For each IP, y, it checks if x is $\psi-$*frequent* in *Stream-Summary*$_y$. If both frequency conditions hold, this pair (x, y) is reported as a possible fraud instance.

7. PERFORMANCE ANALYSIS OF THE SLEUTH ALGORITHMS

Now, we briefly comment on the performance of *2-Pass-SLEUTH* and *1-Pass-SLEUTH* from an analytical perspective. We analyze the accuracy, and the processing time of the SLEUTH algorithms.

7.1 Run Times of the SLEUTH Algorithms

THEOREM 2. 2-Pass-SLEUTH *runs in two passes, with an amortized constant processing time per traffic entry.*

Proof. The proof follows directly from the algorithm definition and the streaming nature of *Space-Saving*. □

We also prove that *1-Pass-SLEUTH* processes each traffic entry in constant amortized time. Hence, *1-Pass-SLEUTH* is effective in a streaming environment.

THEOREM 3. *The* 1-Pass-SLEUTH *algorithm has an amortized constant processing time per traffic entry.*

Proof. The *1-Pass-SLEUTH* algorithm is divided into four main steps. For any pair, (x, y), the first step increments y in *Stream-*

[5]From real network experience, we suggest setting $\epsilon \geq \frac{\varphi}{10} \geq \frac{\phi}{100}$.

```
Algorithm: 1-Pass-SLEUTH(Captured-Correlations(m, n))
begin
  for each pair, (x, y), in the traffic S{
    let x be monitored using Stream-Summary_x
    Space-Saving(Stream-Summary_x, m, y);
    If y turned φ−frequent for x{
      If pair_y exists in IP-Hash-Table{
        IP-Hash-Table.Count_freq(y) + +;
      }else{
        SS_y = new empty Stream-Summary with n counters;
        pair_y = new pair ⟨1, SS_y⟩ ;
        Insert pair_y in IP-Hash-Table;
      }
    }
    let Set^x_infreq be the set of elements turned φ−infrequent
    for each q ∈ Set^x_infreq{
      IP-Hash-Table.Count_freq(q) − −;
      If IP-Hash-Table.Count_freq(q) = 0{
        delete pair_q from IP-Hash-Table;
      }
    }// end for
    If pair_y exists in IP-Hash-Table{
      let y be monitored using Stream-Summary_y
      Space-Saving(Stream-Summary_y, n, x);
    }
  }// end for
end;
```

Figure 7: The *1-Pass-SLEUTH* Algorithm

```
Algorithm: Anomaly-Alarm(Captured-Correlations(m, n))
begin
  for each Stream-Summary_x in Publisher-Hash-Table{
    Integer i = 1;
    while (Count(x, e_i) > ⌈φF(x)⌉ AND i ≤ m){
      let e_i be monitored in IP-Hash-Table at pair_y
      let F''(y) be the total number of y hits in Stream-Summary_y
      If (Count(y, x) > ⌈ψF''(y)⌉){
        output (x, y) as correlated;
      }
      i + +;
    }// end while
  }// end for
end;
```

Figure 8: The *Anomaly-Alarm* Algorithm

Summary$_x$. The second step increments $Count_{freq}(y)$ if y became $\varphi-$*frequent* in the *Stream-Summary*$_x$. The third step decrements $Count_{freq}(q)$, for every q that became $\varphi-$*infrequent* in *Stream-Summary*$_x$. The fourth step increments x in *Stream-Summary*$_y$, if the traffic of y is monitored. From [26], the first and the fourth steps consume amortized constant time. We will show that the second and the third steps also consume amortized constant time.

Any element, q, can only become $\varphi-$*frequent* in a *Stream-Summary* data structure, when q is incremented. Hence, the number of times q is reported as $\varphi-$*frequent* is no more than the number of occurrences of q. Bearing in mind that an element can only become $\varphi-$*infrequent* if it was originally $\varphi-$*frequent*, we deduce that the number of times q is reported as $\varphi-$*infrequent* is no more than the number of occurrences of q. Generalizing that to all the IPs that occur in S_x, for any publisher x, it follows that the total number of times of the elements are reported as frequent or infrequent is no more than $2F(x)$. Dividing this by the size of S_x, then for every hit to x, the average number of IPs reported as frequent or infrequent is at most 2. Since processing counter increments, or decrements, and creating or deleting *Stream-Summary* data structures is done in an amortized constant time because of the hash table, then the second and third steps are done in amortized constant time.

Hence, the four steps have amortized constant time. □

7.2 Space Usages of the SLEUTH Algorithms

From Theorems 4 and 5, *1-Pass-SLEUTH* consumes more space, since $\phi > \varphi$. However, the space requirements of both algorithms are bounded, and can be accommodated on current machines.

THEOREM 4. *The space used by* 2-Pass-SLEUTH *is O($|A_1| * (\frac{1}{\epsilon} + \frac{1}{\phi*\eta})$), where A_1 is the number of publishers.*

Proof. In the first pass, *2-Pass-SLEUTH* keeps a *Publisher-Hash-Table* that contains a *Stream-Summary* data structure for each publisher. In the second pass, for each IP in the *frequent group*, a *Stream-Summary* structure is maintained. The space consumed by *Publisher-Hash-Table* is the size of one *Stream-Summary* structure multiplied by their number. This is equal to *O*($m*|A_1|$) = *O*($\frac{|A_1|}{\epsilon}$).

The maximum number of IPs that can be frequent for one publisher is $\frac{1}{\phi}$. Hence, the maximum possible number of IPs in the *frequent group* is $\frac{|A_1|}{\phi}$. Each of these IPs is allocated a *Stream-Summary* structures requiring up to $n = \frac{1}{\eta}$ counters. Hence, the total space used in the second pass is *O*($\frac{|A_1|}{\phi*\eta}$). □

THEOREM 5. *The space used by* 1-Pass-SLEUTH *is O($|A_1| * (\frac{1}{\epsilon} + \frac{1}{\varphi*\eta})$), where A_1 is the number of publishers.*

Proof. The *Captured-Correlations* data structure used by the *1-Pass-SLEUTH* algorithm consists of two components. The first component is the *Publisher-Hash-Table* that contains *Stream-Summary* data structures for all publishers. The second component is *IP-Hash-Table* that carries the pairs of counters and *Stream-Summary* data structures for IPs.

The space consumed by *Publisher-Hash-Table* is *O*($\frac{|A_1|}{\epsilon}$).

The space consumed by *IP-Hash-Table* is the size of one *Stream-Summary* structure multiplied by their number, since the space used by the hash table and the counters is linear in the number of monitored IPs. The maximum number of IPs that can be frequent for one publisher is $\frac{1}{\varphi}$. Hence, the maximum possible number of IPs' *Stream-Summary* structures is $\frac{|A_1|}{\varphi}$. Each of these structures requires up to $n = \frac{1}{\eta}$ counters. Hence, the total space used by the second component is *O*($\frac{|A_1|}{\varphi*\eta}$). □

7.3 Accuracy of the 1-Pass-SLEUTH Algorithm

Although *1-Pass-SLEUTH* monitors the traffic of IPs approximately, since their traffic is monitored after they are frequent for at least one publisher, practically, its accuracy is very high. Assuming that the traffic characteristics of non-fraudulent publishers and IPs are stable within the analyzed window, for every fraudster, *1-Pass-SLEUTH* guarantees reporting all its exploited IPs as stated in Theorem 6.

Violations of this assumption are expectedly uncommon as the traffic window analyzed is typically an hour or a few hours. Since, fraudsters cannot forcefully influence the traffic of honest sites contracting with the commissioner, they cannot use this assumption to escape *1-Pass-SLEUTH*.

THEOREM 6. *If the characteristics of non-fraudulent traffic are stable within the analyzed window,* 1-Pass-SLEUTH *and* Anomaly-Alarm *report all suspects in the stream.*

Proof. For any fraudulent pair (x, y), let $|S''|$, and $F''(x, y)$ denote the number of hits observed after the creation of *Stream-Summary*$_y$, for all pairs, and for (x, y), respectively. Let N be the size of the entire stream. S_x will consist of alternating *on-segments*, where y is $\phi-$*frequent* in *Stream-Summary*$_x$, and *off-segments*, where y is $\phi-$*infrequent* in *Stream-Summary*$_x$. Since, by the end of S, y is $\phi-$*frequent* for x, then S_x ends in an *on-segment*. During an *on-segment*, the traffic of y is guaranteed to be monitored. Therefore, the traffic of y cannot start being monitored amidst an *on-segment*. That is, just before monitoring the

traffic of y, y was $\phi-$*infrequent* in *Stream-Summary*$_x$, and was $\phi-$*frequent* for x in the portion of S_x after *Stream-Summary*$_y$ was last created, even if this portion contains *off-segments*. Hence, $\frac{F''(x,y)}{F''(x)} > \frac{F(x,y)}{F(x)} > \phi$, where $F''(x)$ is the number of hits observed for x after creating *Stream-Summary*$_y$.

If the traffic characteristics of non-fraudulent pairs do not change drastically within the stream, for any non-fraudulent pair (x, y), the following holds.

$$\frac{|S''|}{N} \approx \frac{F''(x,y)}{F(x,y)} \quad (1)$$

For the rest of the proof, we assume rough equality holds with high probability. Without loss of generality, let x be the only fraudster exploiting y, and y is exploited by x only. Summing Equation 1 for all IPs except y yields $\frac{F''(x,A_2\backslash y)}{F(x,A_2\backslash y)} \approx \frac{|S''|}{N}$. Since $F(x) = F(x,y) + F(x,A_2\backslash y)$, and $F''(x) = F''(x,y) + F''(x,A_2\backslash y)$, then $\frac{F''(x,y)}{F(x,y)} > \frac{F''(x)}{F(x)} = \frac{F''(x,y)+F''(x,A_2\backslash y)}{F(x,y)+F(x,A_2\backslash y)}$. Thus, $\frac{F''(x,y)}{F(x,y)} > \frac{|S''|}{N}$ for this fraudulent pair.

Summing Equation 1 for all sites except x yields $\frac{F''(A_1\backslash x,y)}{F(A_1\backslash x,y)} \approx \frac{|S''|}{N}$. Since $F(y) = F(x,y)+F(A_1\backslash x,y)$, and $F''(y) = F''(x,y)+F''(A_1\backslash x,y)$, then $\frac{F''(x,y)}{F(x,y)} > \frac{F''(y)}{F(y)}$. Since x is $\psi-$*frequent* for y, then $F(x,y) > \psi F(y)$, and hence, $F''(x,y) > \psi F''(y)$. Since $Count(y,x) \geq F''(x,y)$, then $Count(y,x) > \psi F''(x)$.

Thus, y gets reported as $\phi-$*frequent* for x, and x gets reported as $\psi-$*frequent* for y. □

8. EXPERIMENTAL RESULTS

After analyzing the performance and the usability of *2-Pass-SLEUTH* and *1-Pass-SLEUTH*, we report their behavior on the Fastclick network. We experimented with a 45-minute traffic trace of 54,045,873 entries. We evaluated the effectiveness and accuracy of the algorithms. In addition, we explored the tradeoff between the fraud and the detection costs, discussed in § 2.2.

8.1 Algorithmic Assessment

The proposed algorithms were implemented in C++, and were executed on a Pentium IV 2.66 GHz, with 1.0 GB RAM. For *2-Pass-SLEUTH* and *1-Pass-SLEUTH*, in order to measure the *recall* (the number of correct elements found as a percentage of the number of actual correct elements) and the *precision* (the number of correct elements found as a percentage of the entire output), we had to implement an exact solution. The goal is to measure the algorithmic accuracy of the SLEUTH algorithms as compared to an exact algorithm that exactly solves the correlations problem in multidimensional streams. This should not be confused with whether the suspects are fraudulent or not.

We implemented a linked-list representation of the sparse matrix of publishers and IPs, and used it to calculate the correct correlations. However, to process a data sample of size less than 3M entries, the sparse matrix approach ran in over 84 hours.

So, we had to retreat to a smarter exact algorithm. We implemented *Exact*, a single-pass hash-based algorithm that keeps information about the size of the traffic of every publisher-IP pair observed in the dataset. Although *Exact* cannot realistically be used to analyze large datasets, it provides all the correlations when used for smaller datasets. We dissected the traffic dataset by time into 15 data samples that were of roughly equal sizes. Thus, on small datasets, *Exact* makes it possible to measure the *recall* and the *precision*, and assess the savings of the approximate SLEUTH algorithms.

To evaluate the strengths of the SLEUTH algorithms, we ran comprehensive experiments with values of ϕ and ψ varying from 0.1 to 1.0 on a fixed interval of 0.1. For simplicity, we used the same ϕ for all publishers and the same ψ for all IPs. Throughout the experiments, φ was set to $\frac{\phi}{2}$. m was set to $\frac{10}{\phi}$, which guarantees an ϵ of at most $\frac{\phi}{10}$ in the publishers' *Stream-Summary* structures. n was set to $\frac{10}{\psi}$, which guarantees an η of at most $\frac{\psi}{10}$ in the IPs' *Stream-Summary* structures. The results are reported in § 8.1.1. We then examine the online nature of *1-Pass-SLEUTH* by comparing its performance to the single-pass *Exact* as the stream is processed, and report the results in § 8.1.2.

8.1.1 The Effectiveness of the SLEUTH Algorithms

We examined how the algorithms behave as ϕ changes. We issued queries for finding correlations with ψ fixed at 0.1. We varied ϕ from 0.1 to 1.0 on a fixed interval of 0.1. We then examined how the algorithms behave as ψ changes, and varied ψ from 0.1 to 1.0 on a fixed interval of 0.1, with ϕ fixed at 0.1. Every query was issued against the 15 data samples. For various ϕ, the average space, run time, recall, and precision for *2-Pass-SLEUTH*, *1-Pass-SLEUTH*, and *Exact* are plotted in Figures 9(a) through 9(d), respectively. For various ψ, the performance metrics are plotted in Figure 10.

8.1.1.1 The Space and Time Efficiency of SLEUTH.

From Figures 9(a) and 10(a), the space consumptions of the SLEUTH algorithms were at most one fifth that of *Exact* when both ϕ and ψ were 0.1. Since *Exact* keeps complete information, its space usage was constant for all ϕ values. The space consumptions of the SLEUTH algorithms dropped as ϕ increased, since they kept fewer counters in the publishers' *Stream-Summary* structures. In addition, the number of IPs whose traffic was monitored decreased since fewer IPs qualified to be in the *frequent group*. On the other hand, their space usages dropped very slightly as ψ increased, since the SLEUTH algorithms kept fewer counters to monitor the traffic of each monitored IP. However, the number of such monitored IPs did not decrease. This is the reason the decrease in spaces of the SLEUTH algorithms was relatively slower with the increase in ψ than with the increase in ϕ. This is clear when comparing Figures 9(a), and 10(a). The ratio of the space usage of the SLEUTH algorithms to *Exact* dropped from $1:5$ to $1:20$ as ϕ approached 1, while it decreased to $1:6$ as ψ approached 1. On average, *2-Pass-SLEUTH* consumed 10% less space than *1-Pass-SLEUTH*, due to monitoring fewer IPs.

From Figures 9(b) and 10(b), the run times of the SLEUTH algorithms were almost constant. However, they ran 1.5 times faster than *Exact* in Figure 9(b), and 1.2 times faster than *Exact* in Figure 10(b). This is due to fewer increments on the IPs' dimension. On average, *2-Pass-SLEUTH* ran 4% faster than *1-Pass-SLEUTH*, since fewer IPs were monitored as the *frequent group* was identified more accurately. This speed difference shows both algorithms are CPU intensive. The second pass did not slow down *2-Pass-SLEUTH*. Instead, *2-Pass-SLEUTH* benefited from fewer increments on the IPs' dimension.

8.1.1.2 The Accuracy of SLEUTH.

The recall and precision of *2-Pass-SLEUTH* were always constant at 1. The recall of *1-Pass-SLEUTH* varied between 1 and 0.9997 throughout the experiments. From Figure 9(d), the precision of *1-Pass-SLEUTH* decreased slightly from 0.97 to 0.95 as ϕ increased from 0.1 to 1.0. When varying ψ, from Figure 10(d), the precision of *1-Pass-SLEUTH* decreased slightly from 0.97 to 0.91.

8.1.1.3 Comments on the SLEUTH Effectiveness.

When compared to *Exact*, *2-Pass-SLEUTH* runs faster, consumes less space, and gives exact results. However, *2-Pass-SLEUTH* can only start the second pass after the end of the window. The second pass consumed 18% of its run time on average, which is consider-

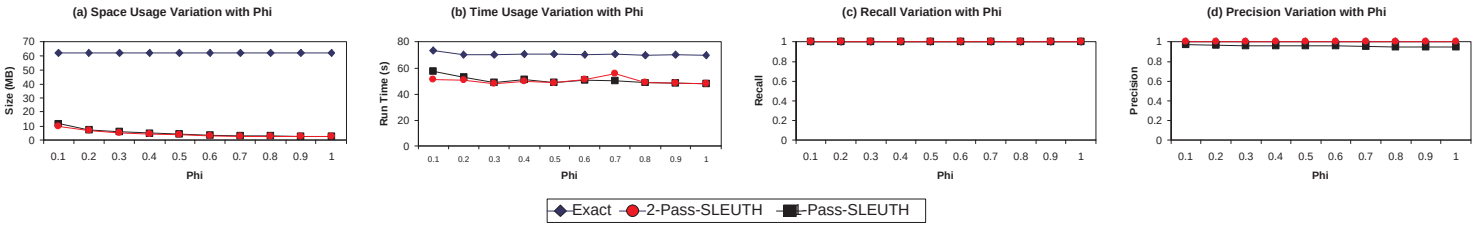

Figure 9: Performance Variation as ϕ Changes Using Real Data

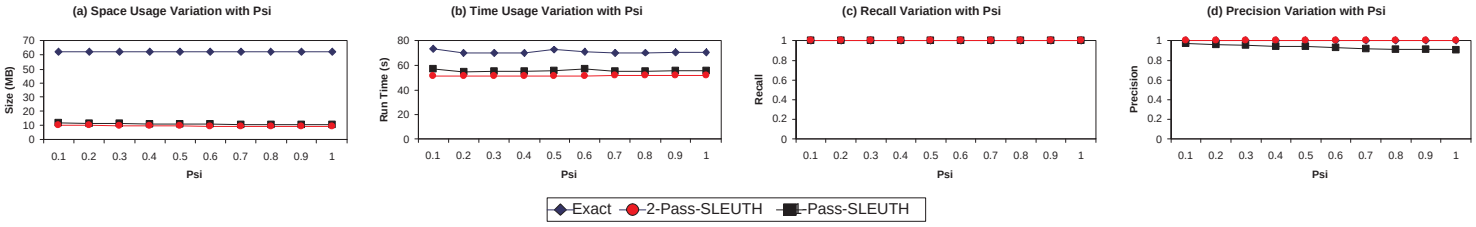

Figure 10: Performance Variation as ψ Changes Using Real Data

able latency in reporting suspects, especially when analyzing long traffic windows, like days. This supports the design decision at Fastclick to consider only single-pass schemes.

When compared to the single-pass *Exact*, *1-Pass-SLEUTH* gives high quality results, uses a lot less space, and runs faster. However, it could be argued that the difference in space can be counteracted by cheap memory. The real strength of *1-Pass-SLEUTH* is clearer when handling data of realistic sizes at a real time rate. The set of experiments presented above handles a very small dataset. On larger datasets, as explained in § 8.1.2, *1-Pass-SLEUTH* consumes much less space than *Exact*. The larger space consumed by *Exact* makes it virtually impossible to process realistic datasets in real time, due to thrashing.

8.1.2 1-Pass-SLEUTH Onlineness

To assess the online nature of *1-Pass-SLEUTH*, we had to examine its scalability. The goal is to show that traffic entries are processed continuously at a constant rate regardless of the window size. We conducted another set of experiments. We issued one query with $\phi = \psi = 0.1$ on the complete traffic stream (54,045,873 entries). We measured the time and the space usages for the algorithms after every 2,000,000 entries, and plotted the results in Figure 12. Clearly, *Exact* could not process the entire dataset due to thrashing.

From Figure 12(a), *Exact* started thrashing after consuming 12M entries. Meanwhile, *1-Pass-SLEUTH* only used 24MB and its space consumption was almost constant throughout the stream. The flat curve of *1-Pass-SLEUTH* is a manifestation of its constant space usage that is a lot smaller than *Exact*.

In terms of execution time (Figure 12(b)), *Exact* thrashed while *1-Pass-SLEUTH* had a constant processing time per entry regardless of the window size, as is clear from its linear trend, until it finished after 19.65 minutes. The fact that it was able to process the traffic the commissioner received in 45 minutes (approximately 54M entry), in less than 20 minutes proves its ability to discover fraud in real time without any lag, and with the ability to report suspects with almost no query latency.

8.2 Comments on Findings

To evaluate the tradeoff between the weakness of the sought correlations and the number of honest suspects (false positives), we ran comprehensive experiments with various values of ϕ and ψ, and sketched the results in Figure 11. Large values of ϕ and ψ produce fewer correlations, fewer honest suspects, and hence, lower cost of fraud detection. However, to look for fraudsters that are

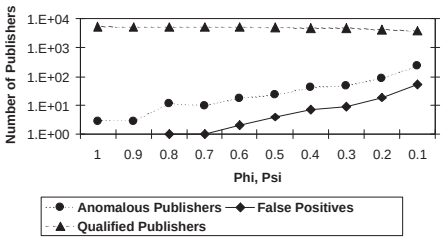

Figure 11: Variation of Percentage of Correlations with ϕ and ψ (Number of Publishers axis has logarithmic scale)

using numerous IPs, and forging slim traffic from each IP to stay under the radar level, the thresholds on ϕ and ψ were lowered. Although some honest suspects get reported, which raises the cost of fraud detection, the SLEUTH algorithms reduces the suspects dramatically compared to the case where they are not used.

For simplicity, we issued queries with $\phi = \psi$. We varied them from 1.0 to 0.1 on a fixed interval of 0.1. For every run, when analyzing the traffic, we only considered publishers receiving at least $\frac{10}{\psi}$ hits, in order to reduce the noise of under-sampled publishers. The number of qualified publishers varied between 3.7k and 5.2k publishers, which is big enough to ensure the validity of the results. We comment on our findings, including validating Assumption 1, and report some statistics on the discovered fraudsters.

8.2.1 Assumption Validation

The problem of detecting single-publisher attacks was modeled as searching for correlations violating Assumption 1. Assumption 1 states that the two dimensions of publishers and IPs are almost independent. That is, there should be no strong correlation between x and y for any observed pair (x, y). Through analysis of real data, we show these correlations are very infrequent. We measured the number of publishers reported as suspects.

From Figure 11, the percentage of the publishers reported increased from 0.06% of the qualified publishers, when ϕ and ψ were 1.0, to 6.37%, when ϕ and ψ were 0.1. The median percentage of qualified publishers reported as suspects, even for combinations of ϕ and ψ not shown in Figure 11, was 0.42%. § 8.2.2 reports very few honest suspects to not only show the validity of Assumption 1, but also the high correlation between fraudsters and exploited IPs.

8.2.2 Tradeoff between Fraud and Detection Costs

Figure 11 is a manifestation of the tradeoff between the costs of the fraud and the detection mechanisms presented in § 2.2. As the

number of suspects increased, the percentage of honest suspects also increased from 0% to 23.18% of the reported suspects (0% to 1.48% of the qualified publishers). The percentage of honest suspects was 0% when the ϕ and ψ were both more than 0.8. The median of honest suspects, even for combinations of ϕ and ψ not shown in Figure 11, was 14.25%.

A suspect was judged to be fraudulent or a false positive based primarily on the sales volume generated by its traffic. The sales information of around 52% of the advertisers contracting with Fastclick were available to judge the value of the traffic from each publisher. This is a good sample to judge all publishers since each publisher generates traffic to almost all the advertisers.

However, the sales information alone is not enough to judge publishers due to the lack of evidence of malicious behavior. Around 17.2% of the publishers had meager sales. Discarding all publishers with sparse sales underpays the commissioner for large honest traffic volumes, as discussed in § 2.2.

8.2.3 Comments on Discovered Frauds

Around 79.4% of the frauds discovered were from 8 IPs or fewer. Around 52.3% of which, used more than a single cookie for their attacks. This shows the proliferation of *NAT-Masquerading*.

It is also worth reporting that among our false positives, we discovered some publishers that were weakly correlated with a few IPs, such that the traffic of the IPs did not yield any sales. However, these publishers were generating sales from traffic from the rest of the IPs. We expect this correlation to be a result of either DDoS attacks; or as attacks of competitors of these publishers that aim at judging these honest suspects as fraudsters to terminate their contracts with commissioners. We discounted the traffic coming from these suspected IPs, but charged the advertisers for the traffic coming from the rest of the IPs.

9. RELATED WORK

After describing our single-publisher attack detection approach and evaluating it on Fastclick network, we summarize the related work to clarify the relevance of the considered problem. Click fraud has been a concern to commissioners since their conception [35][6]. Classical fraud detection judges publishers based on metrics of advertisements on their sites, such as how the ratio of impressions to clicks (the *Click Through Rate*) for advertisements differs from the norms [18]. The classical approach depends on the commissioners' edge in knowing the network-wide behavior of each advertisement. Supposedly, this knowledge is only available to commissioners, since the traffic model conceals from the publishers the advertisements loaded and clicked on their sites.

However, the classical approach can be fooled by fraudsters who can take advantage of a specific site architecture that reveals the advertisements loaded and clicked [29]. This architecture is used by some trusted publishers, like CNN, to maximize revenue among several commissioners. Specifically, CNN redirects advertising traffic to a pool of servers operated by CNN before reaching the commissioners. This architecture reveals the advertisements loaded and clicked by surfers. If adopted by fraudsters, this architecture allows sampling the natural advertisements' behavior. Hence, fraudsters can automate traffic whose metrics comply with real traffic.

Furthermore, the classical approach cannot detect malicious intentions, and is designed to discard low-quality traffic, even if it is legitimate, where the quality of a click or an impression is an estimate of the probability it yields a sale. Aggressively discounting low-quality traffic whose metrics (e.g. CTR) deviate from the norms, or that yields no sales "underpays" honest publishers and commissioners for massive legitimate traffic delivered by their servers.

The main challenge is to distinguish fraudulent traffic from normal traffic. Asking for the cooperation of surfers, as proposed by the cryptographic approaches [3, 30], entails changing the advertising network model to be non-transparent to all surfers, which is unscalable. Moreover, they require the commissioners to uniquely identify surfers, which compromises surfers' privacy.

Analyzing the traffic data, which alleviates the drawbacks of both the cryptographic and classical approaches, was first proposed in [25]. Since commissioners face the dilemma of preserving surfers' privacy versus detecting fraud, they can only perform traffic-mining techniques on aggregate data using temporary surfers' identification, cookie IDs and IP addresses. This identification is temporary and stores no personal identification to preserve surfers' privacy. Meanwhile, machines used in the attacks are still identified enough for satisfactory fraud detection.

Detecting click fraud attacks has recently attracted much attention. In [25], we proposed a simple Bloom filter-based [2] algorithm that detects a naïve click fraud attack, where the publisher runs a script that continuously loads its page and simulates clicks on the advertisements in the page. The solution detects duplicates in a stream of impressions or clicks within a short period of time, like an hour. Experiments on real data were revealing. One of the advertisements was clicked 10,781 times by the same cookie ID in one day.

A more sophisticated click fraud attack was identified by Anupam *et al.* in [1]. It involves a coalition of dishonest publishers. In [27], a solution was proposed for detecting this attack via cooperation between commissioners and Internet Service Providers (ISPs) through identifying associations in a stream of HTTP requests. [28] proposed a generalized coalition attacks detection mechanism by discovering sites that have similar traffic. Our real-data analysis in [28] shows that legitimate sites have highly dissimilar traffic. Sites that receive their traffic from highly similar sets of IPs are almost always suspicious. In [28], we modeled the problem of detecting fraud coalitions in terms of the set similarity problem. We first proposed our *Similarity-Seeker* algorithm that uncovers coalitions of site pairs. We then extended the detection algorithm to detect coalitions of any size by finding all maximal cliques in a sites' similarity graph. On the Fastclick network, 93% of the detected sites were provably fraudsters.

Interestingly, data analysis techniques, like [25, 27, 28], can identify specific patterns and correlations that characterizes fraudulent traffic [23]. Hence, this approach can reveal malicious intentions. Thus, it complements the classical tools that detect low-quality traffic and cannot distinguish it from fraudulent traffic.

Detecting single-publisher attacks complements our algorithms for *coalition* attacks [28] that involve coalitions between several fraudsters. Detecting these two classes of attacks leaves no chances for fraudsters to escape.

10. CONCLUSION AND FUTURE WORK

This paper explored the association between the sophistication of single-publisher attacks, and the difficulty of detecting such attacks; as well as the tradeoff between the single-publisher fraud cost and fraud detection costs. We modeled discovering single-publisher attacks as finding correlations in multidimensional datasets. We devised the approximate *2-Pass-SLEUTH* and *1-Pass-SLEUTH* algorithms to solve the problem efficiently.

We demonstrated the effectiveness and accuracy of the proposed SLEUTH algorithms analytically. Our analytical evaluation was verified through comprehensive experiments on the Fastclick net-

[6]The complementary problem of *hit shaving*, where advertisers do not pay commission on some traffic, was addressed in [33]. The problem of dishonest commissioners has been studied in [21].

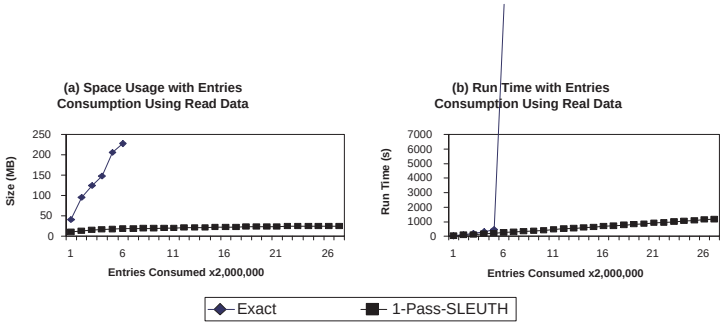

Figure 12: Scalability Using Real Data

work. The online nature of *1-Pass-SLEUTH* and its independence of the stream window size was also established.

Although using the proposed framework, it is difficult to detect attacks from tens of thousands of IPs with a manageable ratio of honest suspects, attacks from tens and hundreds of IPs can still be detected with very few honest suspects. The good news is two folds. First, the number of honest suspects (false positives) is very limited. Interestingly, among the reported correlations, manual investigations detected numerous fraudsters whose contracts were terminated. Second, such attacks are the majority of the attacks due to the difficulty of using tens of thousands of IPs in attacks. Hence, the false positives are also expected to be very few.

Publishing this technique of fraud detection does not help fraudsters escaping detection. To reduce this correlation between the fraudsters' sites and the attacking machines, fraudsters can share traffic with each other to reduce the correlation between their sites and the exploited IPs. This is considered as a *coalition* attack, which has been successfully addressed in [28] by detecting attacks with traffic derived from similar sets of IPs. This work makes it very difficult for fraudsters to launch attacks from a small number of IPs. Hence, it increases the cost or the skills needed to launch single-publisher attacks, with a negligible cost on the commissioner's side.

Acknowledgment

We thank Dr. Jerry Qi Zheng for his useful discussions, and for helping us with deploying *1-Pass-SLEUTH* on Fastclick network.

11. REFERENCES

[1] V. Anupam, A. Mayer, K. Nissim, B. Pinkas, and M. Reiter. On the Security of Pay-Per-Click and Other Web Advertising Schemes. In *Proceedings of the 8th WWW International Conference on World Wide Web*, pages 1091–1100, 1999.

[2] B. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] C. Blundo and S. Cimato. SAWM: A Tool for Secure and Authenticated Web Metering. In *Proceedings of the 14th ACM SEKE International Conference on Software Engineering and Knowledge Engineering*, pages 641–648, 2002.

[4] A. Broder. Data Mining, the Internet, and Privacy. In *Proceedings of the 1st WEBKDD International Workshop on Web Usage Analysis and User Profiling*, pages 56–73, 1999.

[5] CERT Coordination Center. CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks. http://www.cert.org/advisories/CA-1996-21.html, September 19 1996.

[6] M. Charikar, K. Chen, and M. Farach-Colton. Finding Frequent Items in Data Streams. In *Proceedings of the 29th ICALP International Colloquium on Automata, Languages and Programming*, pages 693–703, 2002.

[7] E. Cooke, F. Jahanian, and D. McPherson. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. In *Proceedings of the 1st USENIX SRUTI Workshop on Steps to Reducing Unwanted Traffic on the Internet*, 2005.

[8] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding Hierarchical Heavy Hitters in Data Streams. In *Proceedings of the 29th VLDB International Conference on Very Large Data Bases*, pages 464–475, 2003.

[9] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Diamond in the Rough: Finding Hierarchical Heavy Hitters in Multi-Dimensional Data. In *Proceedings of the 23rd ACM SIGMOD International Conference on Management of Data*, pages 155–166, 2004. An extended version appeared in ACM TKDD Transactions on Knowledge Discovery from Data, 1(4), 2008.

[10] G. Cormode and S. Muthukrishnan. What's Hot and What's Not: Tracking Most Frequent Items Dynamically. In *Proceedings of the 22nd ACM PODS Symposium on Principles of Database Systems*, pages 296–306, 2003. An extended version appeared in the ACM TOCS Transactions on Computer Systems, 30(1): 249-278, 2005.

[11] E. Demaine, A. López-Ortiz, and J. Munro. Frequency Estimation of Internet Packet Streams with Limited Space. In *Proceedings of the 10th ESA European Symposium on Algorithms*, pages 348–360, 2002.

[12] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice. *ACM TOCS Transactions on Computer Systems*, 21(3):270–313, 2003.

[13] S. Goo. 'Click Fraud' Threatens Foundation of Web Ads. *Washington Post Magazine*, October 22 2006.

[14] M. Jakobsson, P. MacKenzie, and J. Stern. Secure and Lightweight Advertising on the Web. In *Proceedings of the 8th WWW International Conference on World Wide Web*, pages 1101–1109, 1999.

[15] C. Jin, W. Qian, C. Sha, J. Yu, and A. Zhou. Dynamically Maintaining Frequent Items over a Data Stream. In *Proceedings of the 12th ACM CIKM International Conference on Information and Knowledge Management*, pages 287–294, 2003.

[16] R. Karp, S. Shenker, and C. Papadimitriou. A Simple Algorithm for Finding Frequent Elements in Streams and Bags. *ACM TODS Transactions on Database Systems*, 28(1):51–55, 2003.

[17] J. Kerkhofs, K. Vanhoof, and D. Pannemans. Web Usage Mining on Proxy Servers: A Case Study. In *Proceedings of the ECML/PKD Workshop on Data Mining for Marketing Applications*, 2001.

[18] D. Klein. Defending Against the Wily Surfer-Web-based Attacks and Defenses. In *Proceedings of the 1st USENIX ID Workshop on Intrusion Detection and Network Monitoring*, pages 81–92, 1999.

[19] M. Liedtke. Google to Pay $90M in 'Click Fraud' Case. *Washington Post Magazine*, March 9 2006.

[20] M. Liedtke. Yahoo Settles 'Click Fraud' Lawsuit. *MSNBC News*, June 28 2006.

[21] S. Majumdar, D. Kulkarni, and C. Ravishankar. Addressing Click Fraud in Content Delivery Systems. In *Proceedings of the 26th IEEE INFOCOM International Conference on Computer Communications*, pages 240–248, 2007.

[22] G. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *Proceedings of the 28th VLDB International Conference on Very Large Data Bases*, pages 346–357, 2002.

[23] C. Mann. How Click Fraud Could Swallow the Internet. *Wired Magazine*, January 2006.

[24] N. Mason. Web Analytics: Insights From the Front Line, Part 2. *ClickZ News*, Feb 5 2008.

[25] A. Metwally, D. Agrawal, and A. El Abbadi. Duplicate Detection in Click Streams. In *Proceedings of the 14th WWW International World Wide Web Conference*, pages 12–21, 2005.

[26] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient Computation of Frequent and Top-k Elements in Data Streams. In *Proceedings of the 10th ICDT International Conference on Database Theory*, pages 398–412, 2005. An extended version appeared in ACM TODS Transactions On Database Systems, 31(3):1095–1133, 2006.

[27] A. Metwally, D. Agrawal, and A. El Abbadi. Using Association Rules for Fraud Detection in Web Advertising Networks. In *Proceedings of the 31st VLDB International Conference on Very Large Data Bases*, pages 169–180, 2005.

[28] A. Metwally, D. Agrawal, and A. El Abbadi. DETECTIVES: DETEcting Coalition hiT Inflation attacks in adVertising nEtworks Streams. In *Proceedings of the 16th WWW International World Wide Web Conference*, pages 241–250, 2007.

[29] A. Metwally, D. Agrawal, A. El Abbadi, and Q. Zheng. On Hit Inflation Techniques and Detection in Streams of Web Advertising Networks. In *Proceedings of the 27th IEEE ICDCS International Conference on Distributed Computing*, 2007.

[30] M. Naor and B. Pinkas. Secure and Efficient Metering. In *Proceedings EUROCRYPT International Conference on the Theory and Application of Cryptographic Techniques*, pages 576–590, 1998.

[31] S. Olsen. Click Fraud Roils Search Advertisers. *CNET News*, March 4 2005.

[32] U. Pooch and A. Nieder. A Survey of Indexing Techniques for Sparse Matrices. *ACM Computing Surveys*, 5(2):109–133, 1973.

[33] M. Reiter, V. Anupam, and A. Mayer. Detecting Hit-Shaving in Click-Through Payment Schemes. In *Proceedings of the 3rd USENIX Workshop on Electronic Commerce*, pages 155–166, 1998.

[34] D. Vise. Clicking To Steal. *Washington Post Magazine*, page F01, April 17 2005.

[35] T. Zeller Jr. With Each Technology Advance, a Scourge. *The New York Times*, October 18 2004.