DETECTIVES: DETEcting Coalition hiT Inflation attacks in adVertising nEtworks Streams *

Ahmed Metwally † Divyakant Agrawal Amr El Abbadi
Department of Computer Science,
University of California at Santa Barbara
Santa Barbara, CA 93106
{metwally, agrawal, amr}@cs.ucsb.edu

ABSTRACT

Click fraud is jeopardizing the industry of Internet advertising. Internet advertising is crucial for the thriving of the entire Internet, since it allows producers to advertise their products, and hence contributes to the well being of e-commerce. Moreover, advertising supports the intellectual value of the Internet by covering the running expenses of publishing content. Some content publishers are dishonest, and use automation to generate traffic to defraud the advertisers. Similarly, some advertisers automate clicks on the advertisements of their competitors to deplete their competitors' advertising budgets. This paper describes the advertising network model, and focuses on the most sophisticated type of fraud, which involves coalitions among fraudsters. We build on several published theoretical results to devise the *Similarity-Seeker* algorithm that discovers coalitions made by pairs of fraudsters. We then generalize the solution to coalitions of arbitrary sizes. Before deploying our system on a real network, we conducted comprehensive experiments on data samples for proof of concept. The results were very accurate. We detected several coalitions, formed using various techniques, and spanning numerous sites. This reveals the generality of our model and approach.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*; K.4.4 [**Computers and Society**]: Electronic Commerce—*Payment schemes; Security*

General Terms

Algorithms, Experimentation, Performance, Security

Keywords

Click Spam Detection, Coalition Fraud Attacks, Approximate Set Similarity, Similarity-Sensitive Sampling, Cliques Enumeration, Real Data Experiments

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

†Part of this work was done while the first author was at FastClick, Inc., a ValueClick company.

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

1. INTRODUCTION

Internet advertising flourishes as the ideal choice for both small and large businesses to target their marketing campaigns to the appropriate customer body on the fly. An Internet advertiser, e.g. ebay, provides an advertising commissioner, e.g. ValueClick, with its advertisements, allocates a budget, and sets a commission for each customer action, such as clicking an advertisement, bidding in an auction, or making a purchase. The Internet publishers, e.g. myspace.com, motivated by the commission paid by the advertisers, contract with the commissioner to display advertisements on their Web sites. The main orchestrators in this setting are the commissioners, who are the brokers between publishers and advertisers, and whose servers are the backstage for targeting and budgeting.

Whenever a surfer visits a publisher's site, the surfer is referred to one of the servers of the commissioner, which picks an advertisement, and embeds it in the publisher's site on the surfer's Browser. If the surfer clicks the advertisement on the publisher's site, the surfer is referred to the commissioner, who logs the click for accounting purposes, and *clicks-through* the surfer to the advertiser's site.

Since publishers earn revenue on impressions (rendering advertisements), as well as clicks they drive to advertisers, there is an incentive for dishonest publishers to *inflate* the number of impressions and clicks their sites generate [3, 5, 25, 28, 34, 35, 38, 45]. Moreover, dishonest advertisers simulate clicks on the advertisements of their competitors to deplete their advertising budgets [32, 39], which limits the exposure of their competitors' advertisements. Fraudulent traffic results in bad reputation for the commissioner, and sometimes in paying forfeitures for advertisers [29, 30]. *Hit inflation* fraud jeopardizes not only the Internet advertising industry, but rather the entire Internet [32].

Hit inflation has been a concern to advertising commissioners since their conception [47][1]. Most of the research investigates publishers' fraud, since the discussion can be generalized to advertisers' fraud. Three main approaches have been proposed to detect *hit inflation* attacks.

The first *classical* fraud detection approach employed a set of tools that judge publishers based on how the advertisements behave on their sites, such as how the ratio of impressions to clicks (the *Click Through Rate*) for each ad-

[1]The complementary problem of advertisers' *hit shaving*, where advertisers do not pay commission on some of the received traffic, was satisfactorily solved in [40].

vertisement differs from the network-wide norms that are supposedly exclusively known for commissioners [28]. However, fraudsters can make use of a specific publishers' site architecture [36], sample the network-wide metrics of advertisements, acquire knowledge about the metrics of the advertisements loaded on their site, and can hence fool the classical tools. In addition, the classical approach cannot detect malicious intentions, and is designed to discard low-quality traffic[2], even if it is legitimate. Aggressively discounting low-quality traffic "underpays" both honest publishers and commissioners for a lot of the traffic their servers deliver.

The second *cryptographic* approach [5, 38] asks for the cooperation of surfers to identify fraudulent traffic. This entails changing the advertising network model to be non-transparent to all surfer, which is unscalable. Moreover, for the solution to be effective, the commissioner has to uniquely identify surfers, which compromises surfers' privacy.

We proposed the third *data analysis* approach in [34]. Data analysis tools perform statistical analysis on aggregate data of surfers' temporary identification, e.g. Cookie IDs and IP addresses. Analysis of cookie IDs and IPs is more privacy friendly than the cryptographic approach. Cookies store no personal information, and they can be blocked, or periodically cleared [33]. An IP is usually assigned to surfers temporarily, and could be shared by several surfers. The philosophy of the *data analysis* approach is not to change the industry model, and to obfuscate individual surfers' identities, but still achieve satisfactory levels of fraud detection.

Data analysis techniques [34, 35] can identify specific patterns that characterizes fraudulent traffic [32]. For instance, to uncover a primitive *hit inflation* attack where a script continuously simulates clicks on advertisements, we proposed a simple Bloom-based [4] algorithm to detect duplicates in a stream of impressions or clicks [34]. Experiments on a real network were revealing. Interestingly, one of the advertisements was clicked 10,781 times by the same cookie ID in one day. Hence, this approach can reveal malicious intentions, and thus complement the classical tools that cannot distinguish low-quality traffic from fraudulent traffic.

The classical and the cryptographic approaches did not distinguish between attacks made by one publisher or a group of publishers forming a coalition. Meanwhile, making this distinction is at the heart of the data analysis approach. By forming coalitions, fraudsters share large pools of attacking machines. Hence, they avoid the cost and overhead of launching highly distributed attacks individually. In addition, *coalition* attacks are more difficult to detect since the patterns of fraudulent traffic are shared by numerous publishers. Therefore, *coalition* attacks can easily circumvent the classical tools. However, the data analysis approach looks for signs of publishers' coalitions. In [35], we devised an algorithm that detects a specific attack [3].

In this paper, we devise a generalized technique that detects all forms of *coalition* attacks. We first build on published theoretical results to devise the *Similarity-Seeker* algorithm that discovers coalitions of size 2. We then extend *Similarity-Seeker* to discover coalitions of arbitrary sizes. Interestingly, when implemented on a real network, *Similarity-Seeker* detected numerous coalitions of various sizes launching attacks with a variety of techniques. The rest of the paper is organized as follows. We start by setting the stage for discovering *coalition* attacks in Section 2. We describe two brute force algorithms in Section 3. Section 4 develops one of the brute force algorithms into *Similarity-Seeker* that discovers coalitions made by pairs of sites. The generalized problem of detecting coalitions of several sites is explored in Section 5. We comment on the findings on a real network in Section 6. We discuss the related work in Section 7, and conclude with our future work in Section 8.

[2]The quality of a click or an impression is an estimate of the probability that it yields a sale.

2. FRAUD DETECTION BY TRAFFIC ANALYSIS

The basic premise of traffic analysis is to draw correlations between the attacking machines, identified by IPs and cookies, and fraudsters. To circumvent the data analysis techniques, fraudulent publishers should dilute the strong correlation between their sites and the machines from which the attacks are launched. This can be done either on the side of the attacking machines or on the side of the attackers' sites. Hence, the spectrum extremes of the *hit inflation* attacks are:

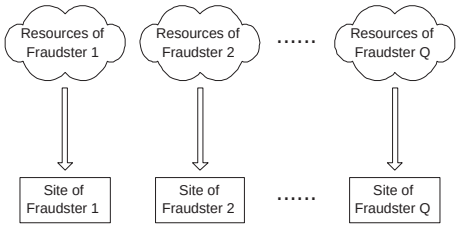
- To dilute the correlation on the machine(s)' side, *non-coalition* attack are performed by one fraudster, while obliterating or frequently changing the identification of the machine(s), or using a huge number of machines.
- To dilute the correlation on the sites' side, *coalition* attack are performed by many fraudsters, where any machine can be used to simulate traffic to any site.

If we detect both *non-coalition* and *coalition* attacks, then any attacker on the spectrum is also detectable. Hence, the attackers will have no leeway around the fraud detection system, and the *hit inflation* problem is ultimately solved.

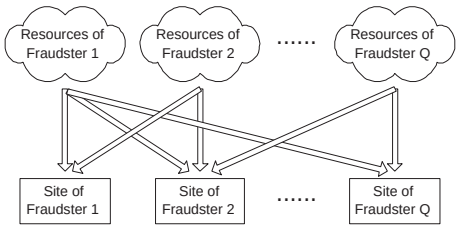
Understandably, the difficulty of detecting a *non-coalition* attack increases as the number of machines from which the attack is launched increases. In its simplest form, launching an attack from one machine, identified by one cookie-ID, can be detected trivially by checking for duplicate impressions and clicks [34]. 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 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. 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®. 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.

This is where the complexity of launching scalable attacks becomes clear. 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



(a) In *non-coalition* attacks, every fraudster generates traffic to its site only.



(b) In *coalition* attacks, every fraudster generates traffic to its and other sites in the coalition.

Figure 1: *Non-Coalition* versus *Coalition* Attacks

to either own the attacking machines, or to control the machines of real surfers through Trojans in order to use the IPs and cookies of real surfers. In other words, launching scalable attacks entails high cost or requires some Trojan-writing skills, since out-of-the-box Trojans are easily detected with anti-virus softwares, and hence are unscalable. Thus, the correlation between the difficulty of launching an attack and the difficulty of detecting it is understandable.

2.1 Forming Fraudsters' Coalitions

To avoid the cost of launching scalable *hit inflation* attacks, fraudsters shift their strategy from launching *distributed non-coalition* attacks, to launching *coalition* attacks where many fraudsters share their resources (machines used in the attack). That is, the machines are used to generate traffic for several sites.

For instance, assume attacker A can generate u hits from each machine it controls, without being rejected by the commissioners' *non-coalition* radars. Another attacker, B, can simulate another u undetectable hits per controlled resource. Assuming no overlap between A's and B's resources, it is more scalable and cost effective to time-share the resources, and generate $2u$ hits for each publisher, instead of doubling the number of resources. The argument can be extended for larger coalitions as illustrated in Figure 1.

Forming coalitions serves two main purposes. The first purpose is to increase the traffic and the revenue while maintaining the same cost per fraudster. The second purpose is to blur the relationship between the identities (IPs and cookie IDs) of the attacking machines and the attackers' sites. Launching *coalition* attacks does not need specialized Web development skills. It only needs minimal resources, and the knowledge of other fraudsters.

For commissioners, detecting *coalition* attacks is challenging since the traffic coming from any attacking machine does not exclusively go to one publisher (Figure 1). Rather, each attacking machine contributes a small portion to the traffic of each fraudster in the coalition. This hinders *non-coalition* radars from associating any publisher with a reasonable amount of resources. Therefore, forming coalitions is the perfect solution for fraudsters to launch scalable attacks from machines whose identifications are weakly correlated with their sites.

2.2 Detecting Fraudsters' Coalitions

To detect *coalition* attacks, the commissioner has to search for publishers' sites with highly similar traffic. A reliable fingerprint of a site's traffic is the set of IP addresses generating the traffic[3]. Given the set of distinct IPs visiting each site, the commissioner has to search for sites whose traffic entries are generated from roughly the same set of IPs[4].

Since the traffic entries of different publishers are interleaved in the log file, the traffic is scanned first to obtain the set of IPs visiting each site, which is then stored in a separate file. For each publisher, A, let $\mathcal{S}_A$ be the set of IPs visiting A. Define $\mathcal{S}_B$ analogously. Several measures of the similarity between $\mathcal{S}_A$ and $\mathcal{S}_B$ exist [16], including the Dice coefficient, $2\frac{|\mathcal{S}_A \cap \mathcal{S}_B|}{|\mathcal{S}_A|+|\mathcal{S}_B|}$; the cosine coefficient, $\frac{|\mathcal{S}_A \cap \mathcal{S}_B|}{\sqrt{|\mathcal{S}_A| \times |\mathcal{S}_B|}}$; and the Jaccard coefficient, $\frac{|\mathcal{S}_A \cap \mathcal{S}_B|}{|\mathcal{S}_A \cup \mathcal{S}_B|}$. The goal is to discover all pairs of sites whose similarity exceeds some threshold, s. Fortunately, as shown in Section 6, any two legitimate sites have negligible similarity.

3. THE BRUTE FORCE ALGORITHMS

We start by describing the two brute force alternatives in Sections 3.1, and 3.2. Since efficiently detecting coalitions is hard, we develop the second algorithms further to detect coalitions of pairs of sites in Section 4. In Section 5, we extend the algorithm to detect coalitions of arbitrary sizes.

3.1 The First Brute Force Algorithm: All-Pairs

The first algorithm considers every possible pair of sites. For every pair, A and B, the *All-Pairs* algorithm calculates any of the three aforementioned similarity coefficients between their sets of IPs by determining the size of the intersection ($\mathcal{S}_A \cap \mathcal{S}_B$) using a sort-merge procedure. Assuming the main memory cannot accommodate the entire traffic, but can accommodate the traffic of any two sites, then the sort-merge procedure can be done in memory. However, the traffic is read from disk $O(D)$ times, where D is the number of publishers' sites. An average-sized commissioner has around 50,000 sites.

The traffic of each site can be presorted. The presorting step is $O(D|\mathcal{S}|\log(|\mathcal{S}|))$, where $\mathcal{S}$ is the largest IP set. The number of distinct IPs visiting a site in a day is around 25,000, but reaches 1,000,000 for some sites. Presorting reduces the total in-memory complexity from $O(D^2|\mathcal{S}|^2)$ to $O(D|\mathcal{S}|(D+\log(|\mathcal{S}|)))$.

[3]Through the sequel, we concentrate on fraud detection, and thus we assume the source IPs of the traffic are not spoofed. Counteracting spoofing has been studied in [15].

[4]To simplify the presentation, we assume all the IPs are equally treated. However, the entire discussion can be trivially generalized for the case where special IPs, such as those coming from a specific location or known to belong to Internet Service Providers (ISPs), are given different weights.

Optimizing All-Pairs.

The *All-Pairs* algorithm can be enhanced by using the triangle inequality. This limits the optimization to the Jaccard coefficient since $1-\frac{|\mathcal{S}_A\cap\mathcal{S}_B|}{|\mathcal{S}_A\cup\mathcal{S}_B|}$, the *Jaccard dissimilarity* of $\mathcal{S}_A$ and $\mathcal{S}_B$, satisfies the triangle inequality. That is, for any three sites A, B, and C, it is true that $\left(1-\frac{|\mathcal{S}_A\cap\mathcal{S}_C|}{|\mathcal{S}_A\cup\mathcal{S}_C|}\right)+\left(1-\frac{|\mathcal{S}_B\cap\mathcal{S}_C|}{|\mathcal{S}_B\cup\mathcal{S}_C|}\right)\geq\left(1-\frac{|\mathcal{S}_A\cap\mathcal{S}_B|}{|\mathcal{S}_A\cup\mathcal{S}_B|}\right)$. The dissimilarity based on the other two coefficients does not satisfy the triangle inequality. The optimization entails looking for a third site, C, such that A and B can be judged similar or dissimilar using the similarity already calculated for the pair A and C, and the pair B and C in a dynamic programming scheme. Thus, when testing A and B for having similarity exceeding the threshold s, if there exists a site, C, such that $(1-s)\geq\left(1-\frac{|\mathcal{S}_A\cap\mathcal{S}_C|}{|\mathcal{S}_A\cup\mathcal{S}_C|}\right)+\left(1-\frac{|\mathcal{S}_B\cap\mathcal{S}_C|}{|\mathcal{S}_B\cup\mathcal{S}_C|}\right)$, then $(1-s)\geq\left(1-\frac{|\mathcal{S}_A\cap\mathcal{S}_B|}{|\mathcal{S}_A\cup\mathcal{S}_B|}\right)$, and A and B are similar.

Conversely, if there exists a site, C, such that $(1-s)\leq\left|\left(1-\frac{|\mathcal{S}_A\cap\mathcal{S}_C|}{|\mathcal{S}_A\cup\mathcal{S}_C|}\right)-\left(1-\frac{|\mathcal{S}_B\cap\mathcal{S}_C|}{|\mathcal{S}_B\cup\mathcal{S}_C|}\right)\right|$, then A and B are dissimilar because $(1-s)\leq\left(1-\frac{|\mathcal{S}_A\cap\mathcal{S}_B|}{|\mathcal{S}_A\cup\mathcal{S}_B|}\right)$.

However, as shown in Section 6, legitimate sites have slim similarity, which makes this optimization ineffective for faster discovery of sites with similar traffic.

3.2 The Second Brute Force Algorithm: All-IPs

Instead of performing a sort-merge for every possible pair of sets of IPs, the *All-IPs* algorithm performs a sort-merge for all the sets together. Hence, all sites sharing a specific IP are identified, and *All-IPs* calculates the similarity of pairs that have a common IP. Counting the number of IPs shared by any pair of sites requires one scan on the sorted data. However, the sort-merge is done out-of-memory[5]. Hence, the traffic is read from disk $O(\log(D))$ times. The first presorting scan has in-memory complexity of $O(D|\mathcal{S}|\log(|\mathcal{S}|))$. Then, $O(\log(D))$ scans are made for merging, each has an in-memory complexity of $O(D|\mathcal{S}|)$, yielding a total in-memory complexity of $O(D|\mathcal{S}|(\log(D)+\log(|\mathcal{S}|)))$.

The *All-IPs* brute force algorithm has been proposed before for measuring set similarity in the context of finding similar files and domains on the Internet [9, 10, 13]. In a collection of D documents, each document is represented by the set of all statements of a specific length the document contains. This is analogous to our problem of finding highly similar sets of IPs among a large collection of sets.

Optimizing All-IPs.

To compact the sets, Broder *et al.* [9, 10, 13] proposed sampling. Reducing the size of $\mathcal{S}$ decreases the computations. Sampling results in the sort-merge producing fewer IPs as well as fewer sites sharing any IP.

However, random sampling from the two sets $\mathcal{S}_A$ and $\mathcal{S}_B$ greatly skews the size of the intersection. In the worst case, random sampling can lead to an empty intersection even though the actual intersection is not empty [17]. Therefore, the sampling technique used should scale down $|\mathcal{S}_A\cap\mathcal{S}_B|$ by the same factor it scales down $|\mathcal{S}_A|+|\mathcal{S}_B|$, $\sqrt{|\mathcal{S}_A|\times|\mathcal{S}_B|}$, or $|\mathcal{S}_A\cup\mathcal{S}_B|$, to conserve the Dice, cosine, or Jaccard coefficients, respectively. Three hash-based sampling techniques were proposed by Broder *et al.* to conserve the similarity. Since our final algorithm is based on the *All-IPs* brute force algorithm, we discuss the three techniques in Section 4.

[5][46] gives a good survey on external memory algorithms.

4. DETECTING COALITIONS OF PAIRS OF SITES

In this section, we develop *All-IPs* to efficiently detect all coalitions of size 2. In Section 4.1, we describe the sampling of IPs visiting sites, and derive the sample size that guarantees a specific error. In Section 4.2, we propose *Similarity-Seeker* for detecting sites with similar traffic.

4.1 Similarity-Sensitive Sampling

The IP sets should be sampled, such that any pair of sites, A and B, whose IP sets, $\mathcal{S}_A$ and $\mathcal{S}_B$, have a similarity exceeding a threshold, s, is discovered with high probability. For any set $\mathcal{S}$ defined on domain $\Omega\subseteq\mathcal{N}$, let $\pi:\Omega\rightarrow\Omega$ be a permutation of Ω chosen uniformly at random; $g:\Omega\rightarrow\mathcal{N}$ be an arbitrary injection; and $MOD_y(\mathcal{S})$ be the subset of the elements that are 0 modulo y. $Y_{y,g,\pi}(\mathcal{S})$, defined as $MOD_y(g(\pi(\mathcal{S})))$, is a similarity-sensitive sample of $\mathcal{S}$. In [9], $\frac{|Y_{y,g,\pi}(\mathcal{S}_A)\cap Y_{y,g,\pi}(\mathcal{S}_B)|}{|Y_{y,g,\pi}(\mathcal{S}_A)\cup Y_{y,g,\pi}(\mathcal{S}_B)|}$ was shown to be an unbiased estimator of the Jaccard coefficient, $\frac{|\mathcal{S}_A\cap\mathcal{S}_B|}{|\mathcal{S}_A\cup\mathcal{S}_B|}$. Although [9, 10, 13] tackled the problem in the context of Jaccard similarity only, we like to point out the generality of the sampling method. It is easy to show $2\frac{|Y_{y,g,\pi}(\mathcal{S}_A)\cap Y_{y,g,\pi}(\mathcal{S}_B)|}{|Y_{y,g,\pi}(\mathcal{S}_A)|+|Y_{y,g,\pi}(\mathcal{S}_B)|}$ is an unbiased estimator of the Dice coefficient, and $\frac{|Y_{y,g,\pi}(\mathcal{S}_A)\cap Y_{y,g,\pi}(\mathcal{S}_B)|}{\sqrt{|Y_{y,g,\pi}(\mathcal{S}_A)|\times|Y_{y,g,\pi}(\mathcal{S}_B)|}}$ is an unbiased estimator of the cosine coefficient

Broder *et al.* proposed other techniques to estimate Jaccard similarity in specific. For any set, $\mathcal{S}$, from a totally ordered domain, let $MIN_z(\mathcal{S})$ be the subset containing the smallest z elements in $\mathcal{S}$ if $|\mathcal{S}|\geq z$; otherwise $MIN_z(\mathcal{S})=\mathcal{S}$. Let $\mathcal{S}_A$ and $\mathcal{S}_B$ be the sets of IPs visiting sites A and B, respectively. Define $Z_{z,\pi}(\mathcal{S}_A)$ as $MIN_z(\pi(\mathcal{S}_A))$, and define $Z_{z,\pi}(\mathcal{S}_B)$ analogously. It was shown in [13] that an unbiased estimator of the Jaccard coefficient of $\mathcal{S}_A$ and $\mathcal{S}_B$ is given by $\frac{|MIN_z(Z_{z,\pi}(\mathcal{S}_A)\cup Z_{z,\pi}(\mathcal{S}_B))\cap Z_{z,\pi}(\mathcal{S}_A)\cap Z_{z,\pi}(\mathcal{S}_B)|}{|MIN_z(Z_{z,\pi}(\mathcal{S}_A)\cup Z_{z,\pi}(\mathcal{S}_B))|}$.

The number of samples generated by the former method, based on $Y_{y,g,\pi}(\mathcal{S})$, grows with $|\mathcal{S}|$, which could be inconvenient; while the number of sample generated by the latter method, based $Z_{z,\pi}(\mathcal{S})$, is fixed. On the other hand, the former method is easier to calculate, and can be used for all similarity coefficients. However, it is difficult to derive any guarantees on the quality of both estimators.

In [10], a MinHash-based Jaccard estimator was adopted. Since this is the only estimator that we can draw error guarantees for, we use it in the sequel. Let π_i be a permutation of Ω chosen uniformly at random. For a set $\mathcal{S}=\{s_1,s_2,\dots\}$ on Ω, let $\min_{\pi_i}(\mathcal{S})$ be $\pi_i^{-1}(\min(\pi_i(s_1),\pi_i(s_2),\dots))$. Therefore, for sets $\mathcal{S}_A$ and $\mathcal{S}_B$, $\min_{\pi_i}(\mathcal{S}_A)=\min_{\pi_i}(\mathcal{S}_B)$ if and only if $\min_{\pi_i}(\mathcal{S}_A\cup\mathcal{S}_B)\in(\mathcal{S}_A\cap\mathcal{S}_B)$. Since π_i is chosen uniformly at random, then all the elements in $(\mathcal{S}_A\cup\mathcal{S}_B)$ are equiprobable to become $\min_{\pi_i}(\mathcal{S}_A\cup\mathcal{S}_B)$. Hence, $\min_{\pi_i}(\mathcal{S}_A)=\min_{\pi_i}(\mathcal{S}_B)$ with probability $\frac{|\mathcal{S}_A\cap\mathcal{S}_B|}{|\mathcal{S}_A\cup\mathcal{S}_B|}$, which is the Jaccard coefficient of $\mathcal{S}_A$ and $\mathcal{S}_B$. Therefore, we can calculate an unbiased estimate of the Jaccard coefficient of the pair $\mathcal{S}_A$ and $\mathcal{S}_B$ as follows. Construct a set of n independent uniform permutations, $\pi_1,\pi_2,\dots,\pi_n$, and calculate $\min_{\pi_i}(\mathcal{S}_A)$ and $\min_{\pi_i}(\mathcal{S}_B)$ for $1\leq i\leq n$. The unbiased estimate is

given by $\frac{Count(i|\min_{\pi_i}(\mathcal{S}_A)=\min_{\pi_i}(\mathcal{S}_B))}{n}$, which is the ratio of permutations (samples) where both minimums agree.

4.1.1 How Big is n? Error Analysis of the Jaccard Estimator.

In advertising networks, having guarantees on the quality of the results is very crucial. The commissioner has to know, with very low error rate, sites whose traffic are highly similar. Discarding sites erroneously reduces the commissioner's revenue, while charging advertisers for fraudulent traffic puts the commissioner at risk. However, no robust error analysis for the estimators in [9, 10, 13] was provided.

Since each permutation generates one sample, to discuss error analysis, we use the two terms interchangeably. We calculate the minimum number of samples, n, for a specific confidence interval estimate on the Jaccard coefficient of $\mathcal{S}_A$ and $\mathcal{S}_B$ using the sampling method proposed in [10]. For each permutation π_i, we model comparing the samples, $\min_{\pi_i}(\mathcal{S}_A)$ and $\min_{\pi_i}(\mathcal{S}_B)$, as an independent Bernoulli trial with unknown success probability, $p = \frac{|\mathcal{S}_A \cap \mathcal{S}_B|}{|\mathcal{S}_A \cup \mathcal{S}_B|}$.

A Bernoulli random variable is 1 with probability p, and is 0 with probability $(1-p)$. Based on n samples, the Maximum Likelihood Estimator of p is given by $\hat{p} = \frac{\sum x_i}{n}$, where the variance of $\hat{p}$ is given by $\frac{p(1-p)}{n}$. Since the distributions of the samples are independent, and identical, the central limit theorem and the law of large numbers state, for large n, $\hat{p}$ is approximately normally distributed. Thus, if K_α denotes the cdf of the standard normal distribution between $-\infty$ and α, then both $\mathbf{Pr}\left(-K_{\alpha/2} < \frac{\hat{p}-p}{\sqrt{p(1-p)/n}} < K_{\alpha/2}\right)$ and $\mathbf{Pr}\left(-K_\alpha < \frac{\hat{p}-p}{\sqrt{p(1-p)/n}}\right)$ are equal to $1-\alpha$.

Therefore, a two-sided $(1-\alpha)$ approximate confidence interval for $\frac{|\mathcal{S}_A \cap \mathcal{S}_B|}{|\mathcal{S}_A \cup \mathcal{S}_B|}$ is given by $\hat{p} \pm K_{\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$, and a one-sided $(1-\alpha)$ approximate confidence interval is given by $\left[\hat{p} - K_\alpha\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}, 1\right]$.

One-sided confidence intervals are more interesting for our application, since we are looking for pairs of sites with Jaccard coefficient exceeding a threshold s. Therefore, the minimum size, n, of the permutations family that would guarantee $(1-\alpha)$ confidence that the estimator exceeds s is $n = \left\lceil \hat{p}(1-\hat{p})\left(\frac{K_\alpha}{\hat{p}-s}\right)^2 \right\rceil$. Clearly, the sample size is not bounded in the general case. However, if p is to be estimated within a margin of error of ϵ, then this bounds n by $\left\lceil \left(\frac{K_\alpha}{2\epsilon}\right)^2 \right\rceil$ [7]. Statistically, this is interpreted as: with probability at least $1-\alpha$, the estimator is no more than ϵ less than the user threshold, s, i.e., $\mathbf{Pr}(\hat{p} > s - \epsilon) \geq 1 - \alpha$. For instance, $n = \left\lceil \left(\frac{1.645}{2\times 0.04}\right)^2 \right\rceil = 423$ permutations guarantees that a pair of sites is estimated to be similar is truly similar within an error of $\epsilon = 0.04$ with probability 0.95.

4.1.2 A Discussion of Permutations.

Random permutations are needed to implement this Jaccard estimator. Representing each truly random permutation requires $|\mathcal{N}|\log|\mathcal{N}|$ bits, where practically $|\mathcal{N}| = 2^{64}$ [24]. This motivated Broder *et al.* to define min-wise independent (MWI) permutations in [11]. $\mathcal{F}$ is a family of MWI permutations on a domain $\Omega \subseteq \mathcal{N}$, if when π is chosen at random from $\mathcal{F}$, then for any $x \in \Omega$, $\mathbf{Pr}(\min(\pi(\Omega)) = \pi(x)) = \frac{1}{|\Omega|}$. That is, all the elements of any domain, Ω, have equal chances to be the minimum element of the image of Ω under any π. However, MWI families are impractical, since the cardinality of any such family is at least $e^{|\Omega| - o(|\Omega|)}$ [11]. In our case, Ω is the IP domain.

```
Procedure: Samples-Select(Integer n)
begin
  //Constructing permutations
  for (Integer i = 1, i ≤ n, i++){
    Construct π_i using the technique in [12];
  }// end for
  //Traffic separation
  for each TrafficEntry e = (Site_ID, IP){
    Write e to the traffic file of Site_ID on disk;
  }
  //Selecting samples
  for each Site_ID{
    IP Site-Samples[n] = empty array;
    for each TrafficEntry e = (Site_ID, IP){
      for (Integer j = 1, j ≤ n, j++){
        if (π_j(IP) < π_j(Site-Samples[j])){
          Site-Samples[j] = IP;
        }
      }// end for
    }// end for
    let Site_ID be stored at row i in Samples
    Write Site-Samples to Samples[i] on disk;
  }// end for
end;
```

Figure 2: The *Samples-Select* Procedure.

To achieve easy computation, Indyk proposed a relaxed version of MWI, $\epsilon-$MWI [24], defined as follows. For any domain $\Omega \subset \mathcal{N}$, and $x \in (\mathcal{N} - \Omega)$, if $\pi : \mathcal{N} \to \mathcal{N}$ is chosen at random from $\mathcal{F}$, then $\mathbf{Pr}(\pi(x) < \min(\pi(\Omega))) = \frac{1\pm\epsilon}{|\Omega|+1}$.

We rewrite this in a simpler form as, if $x \in \Omega$, then $\left|\mathbf{Pr}(\pi(x) = \min(\pi(\Omega))) - \frac{1}{|\Omega|}\right| \leq \frac{\epsilon}{|\Omega|}$. That is, all the elements of any domain, Ω, have *almost* equal chances to be the minimum element of the image of Ω under any π. [24] provides a construction of compact $\epsilon-$MWI permutations.

An even more practical variation of MWI is pair-wise independent (PWI) permutations [11]. $\mathcal{F}$ is a family of PWI permutations if for any $\{s_1, s_2, t_1, t_2\} \subseteq \Omega$, $s_1 \neq s_2$, $t_1 \neq t_2$, if π is chosen at random from $\mathcal{F}$ then $\mathbf{Pr}((\pi(s_1) = t_1) \wedge (\pi(s_2) = t_2)) = \frac{1}{|\Omega|\times(|\Omega|-1)}$. Although [11] showed that PWI families can be viewed as $\epsilon-$MWI families with ϵ as large as $\log\Omega$, in practice, they have many implementations and are widely used [12]. For instance, the performance of linear independence, of the form $\pi_i(x) = a_i x + b_i$ modulo c, is acceptable in real life if $\Omega \subseteq \{1, \ldots, c\}$, $\Omega, c \to \infty$, c is a prime, a_i and b_i are chosen at random, and $a_i \neq 0$ [11, 6]. [12] provided an efficient way to construct a family of $\epsilon-$MWI permutations using a random invertible Boolean matrix, coupled with a redundant input representation, yielding an ϵ of 0.25 in the worst case, and 0.06 in practice. We choose this implementation [12] due to its guaranteed tolerable error.

4.2 The Similarity-Seeker Algorithm

Now, we have a reliable sampling technique that efficiently estimates the similarity of huge sets of IPs visiting sites. Moreover, for a given error bound and confidence, we know the number samples to collect. We use this sampling method to describe the *Similarity-Seeker* algorithm.

The algorithm starts by collecting samples using the technique in [12]. It uses every permutation to discover all pos-

```
Algorithm: Similarity-Seeker(Double s, ϵ, α, Integer l)
begin
  //Calculating the number of samples
  Integer n = ⌈(K_α / 2ϵ)^2⌉;
  //Creating an array of samples
  IP Samples[|Sites|][n] = empty D × n array;
  Samples-Select(n);
  //Allocating space for C
  Set <SiteID, SiteID, Integer> C = empty set;
  for (Integer j = 1, j ≤ n, j++){// IPs loop
    //Allocating space for H
    HashTable <IP, SiteList> H = empty hash table;
    for (Integer i = 1, i ≤ |Sites|, i++){// Sites Loop
      //Populating H with lists of sites sharing an IP
      let Site_ID be the SiteID at Samples[i]
      let IP be the IP at Samples[i, j]
      Insert Site_ID into H[IP].SiteList;
    }// end for
    //Incrementing similarity of sites sharing the j^th IP
    for each SiteList SL in H{
      if (|SL| < l){// Sites do not share a popular IP
        for each two sites, A and B, in SL{
          if ((A, B) ∈ C){
            C[(A, B)].Integer ++;
            if (sn − 1 ≥ C[(A, B)].Integer > sn){
              Output (A, B) as similar;
            }
          } else{
            Insert (A, B, 1) into C;
          }
        }// end for
      }
    }// end for
  }// end for
end;
```

Figure 3: The *Similarity-Seeker* Algorithm.

sible pairs of sites that share an IP in their samples. The algorithm employs three data structures: *Samples*, C, and H. *Samples* is a $D \times n$ array whose rows represent sites, columns represent permutations, and cells represent samples for sites under permutations. The *Samples* array is populated by the *Samples-Select* procedure described next. C is a set of triplets: two site IDs and the number of shared samples. C tracks pairs of sites with shared samples exceeding sn, where s is the similarity threshold. H is a hash table of tuples: a sample ID, and a list of sites sharing it.

4.2.1 Selecting the Samples.

To populate *Samples*, the *Samples-Select*, as sketched in Figure 2, procedure makes two preliminary scans on the traffic data. In the first scan, *Samples-Select* separates the traffic of each site in an individual file that can fit in memory. For each site, *Samples-Select* makes a second pass where it hashes each traffic entry using all the permutations. After collecting all the samples, the file is written back to the right row in *Samples*. We only require the memory to accommodate one row or one column of *Samples* at a time. This increases the scalability of the proposed algorithm. The in-memory complexity of *Samples-Select* is $O(D|\mathcal{S}|n)$.

4.2.2 Discovering Similar Pairs.

Once *Samples* is populated, *Similarity-Seeker* reads it in a column-major manner. For every column (permutation), a hash table, H, is temporarily constructed for holding D samples and their corresponding lists of sites sharing each sample. The hash table is populated after one scan on the column, and the list of sites sharing each IP is populated. A scan is then made on H. Any list that contains a large number, l, of sites sharing a sample is discarded. This sample is probably the IP address of an Internet Service Provider (ISP) or a Network Address Translation (NAT) box, that is shared by hundreds of computers. Otherwise, for each pair in a list sharing a sample, a corresponding element is created with a number of shared samples of 1, and is inserted in C, if it does not already belong to C, or is incremented otherwise. At any time, if the incremented pair satisfies the similarity s, *Similarity-Seeker* outputs this pair as similar. The algorithm for discovering all pairs with a similarity exceeding s with confidence α and error ϵ is presented in Figure 3.

4.2.3 The Similarity-Seeker Complexity.

In addition to the two scans on the traffic date made by *Samples-Select*, *Similarity-Seeker* makes only one scan on *Samples*. Since *Samples* is typically smaller than the entire traffic, then the traffic is considered to be scanned only thrice, as compared to the $O(\log(D))$ in the *All-IPs* algorithm, where D is around 50,000. Calculating the in-memory complexity of *Similarity-Seeker* is more involved. To establish a bound on the in-memory complexity of processing one column by *Similarity-Seeker*, we have to consider two extreme cases. The first is where $l-1$ sites share the same sample, and all the other $\frac{D-(l-1)}{2}$ samples are shared by two sites. The second is where all the D samples are clustered to form lists of length $l-1$. Then the complexity of processing one column is $O\big(\max\big(l^2+D, lD\big)\big)$. The complexity of any non-extreme case is clearly a linear combination of $O\big(l^2+D\big)$ and $O(lD)$. Since the lD factor dominates, and there are n columns, the total complexity of *Similarity-Seeker* is $O\big(\frac{lD}{\epsilon^2}\big)$ with a tiny hidden constant. In practice, the complexity is less due to the dissimilarity of IPs visiting sites, and hence less clustering of sites. We examine the relationship between the estimated sites' similarity and the parameter l in Section 6. Interestingly, some site pairs retain their similarity, no matter how small l is set.

Due to the smaller number of sets, D, in our case (50,000 instead of all the Internet documents) we assumed that all the D samples from one permutation can fit in memory. Thus *Similarity-Seeker* avoids the out-of-memory sort-merge performed by *All-IPs* with all the associated I/O and computational overheads.

5. DETECTING COALITIONS OF ARBITRARY SIZES

Similarity-Seeker can efficiently detect coalitions of pairs of sites. However, as we discuss in Section 5.1, attackers form coalitions of sizes exceeding 2 to stay under the radar level by giving up some greed. Hence, we extend *Similarity-Seeker* to detect larger coalitions in Sections 5.2, and 5.3.

5.1 Greed versus Subtleness Given a Coalition Size

A group of attackers, of size Q, can make hard-to-detect coalitions by not sharing all the resources together. Instead, each publisher would share each resource it controls with only q random attackers. Hence, as shown by Theorem 1 the pairs' similarities drop, while still gaining from coalitions.

THEOREM 1. *A group of attacking publishers, of size Q, where each publisher shares each resource it controls with only $q < Q$ random publishers, reduces similarity between pairs from* 1 *to* $\frac{q(q+1)}{2(Q-1)+q(2Q-q-3)}$ *and achieves a gain of* q.

Proof. Assume each site in the attacking group controls r resources, the Jaccard coefficient is used for similarity, and that a resource shared by A with B is not re-shared by B. For any two sites, A and B, $\mathcal{S}_A \cap \mathcal{S}_B$ is given by the resources shared by A with B, the resources shared by B with A, and the resources shared by all the other $Q-2$ nodes with both A and B. This is equal to $2r\frac{q}{Q-1} + r(Q-2) \times \frac{(q)(q-1)}{(Q-1)(Q-2)} = r(q+1)\frac{q}{Q-1}$. $\mathcal{S}_A \cup \mathcal{S}_B$ is given by the resources controlled by A, the resources controlled by B, and the resources shared by all the other $Q-2$ nodes with either A or B. This is equal to $2r + 2r(Q-2)\frac{q}{Q-1} - r(Q-2) \times \frac{(q)(q-1)}{(Q-1)(Q-2)}$. Hence the Jaccard coefficient is given by $\frac{q(q+1)}{2(Q-1)+q(2Q-q-3)}$. The resources directing traffic to each site is $r(q+1)$, after forming the coalition, instead of the r resources controlled by each site. □

COROLLARY 1. *By increasing the size of a coalition, attackers can sustain similarity between pairs at a low level, while still increasing their gains.*

Proof. To keep pairs' similarity below a detectable level, s, the gain of each publisher is q, where q is given by $\frac{1}{2(s+1)} \times \left((2Q-3)s - 1 + \sqrt{(2Qs+1)^2 - (4Qs^2+1) + (1-s)^2}\right)$.

Hence, similarity between pairs can be sustained at any level, s, while increasing the gain, q, by forming larger groups, i.e. increasing Q. □

From Theorem 1, if $q > 1$ and $q = \sqrt{Q}$, the Jaccard coefficient is less than $\frac{1}{\sqrt{Q}}$. Fortunately, as shown in Section 6, any two legitimate sites have negligible similarity. Therefore, even if attackers form large coalitions, the subtle similarity between pairs is still above the norm, and is still detectable by *Similarity-Seeker*.

5.2 Detecting Large Coalitions based on Pairs of Sites

Increasing the size of coalitions from pairs to large groups shifts our focus from searching for pairs of sites with highly similar traffic to searching for groups with moderate similarity. For two main reasons, the solution in Section 4 has to be extended for coalitions of arbitrary sizes. First, outputting one set of several sites, such that each pair of sites have similar traffic, establishes the evidence for the fraudsters' malicious intention. It is very unlikely, for any random group of sites of size Q, that all possible pairs are similar. If any two publishers' sites can be mistakenly judged to have similar traffic with probability ρ. Then, mistakenly judging that Q random sites are involved in a *coalition* attack has a probability of $\rho^{\frac{Q(Q-1)}{2}}$. For instance, if $\rho = 0.1$, then erroneously judging a small group of 5 random sites to be in coalition has a probability of 10^{-10}.

Second, outputting one set of several sites is more concise than outputting all the possible pairs in that set. For instance, if 3 coalitions of size 50 fraudsters are discovered, it is more convenient for the management to verify a list of 3 entries, each of size 50, than to examine $3 \times \frac{50\times 49}{2} = 3225$ entries, each of size 2. In addition, the output conciseness gives a more panoramic picture of the coalition and facilitates manual investigations like checking for common features of the sites, such as contract date, and earning rate.

Therefore, we need to condense the detected pairs of sites into groups of sites. sites' similarity can be modeled as an undirected graph whose nodes represent sites, and whose edges connect pairs of sites with similar traffic. The objective is to search for, instead of just edges, all maximal cliques in this huge graph.

5.3 Discovering All Maximal Cliques in the Sites' Similarity Graph

Let $G = (V, E)$ be the sites' similarity graph, where the set of nodes V represent sites, and E is a set of edges connecting pairs of sites if and only if their similarity is at least s. We will be interchangeably referring to sites and nodes representing them.

For a subset $W \subseteq V$, $G(W) = (W, E(W))$ with $E(W) = \{(v,w) \in W \times W | (v,w) \in E\}$ is called a *subgraph* of G induced by W, and G is called a *supergraph* of $G(W)$. $G(W)$ is said to be a *clique* if and only if $(v,w) \in E \ \forall \ v,w \in W$. $G(W)$ is a *maximal clique* if it is not a *proper subgraph* of another *clique*. The objective is to find all *maximal clique* in the sites' similarity graph G. The graph $\bar{G} = (V, \bar{E})$ is a complementary graph of G if $\bar{E} = \{(v,w) \in V \times V | (v,w) \notin E\}$. A subset $W \subseteq V$ is an *independent set* if and only if $(v,w) \notin E \ \forall \ v,w \in W$. W is a *maximal independent set* if it is not a *proper subgraph* of another *independent set*. Finding all *maximal clique* in a graph G is equivalent to finding all *maximal independent sets* in $\bar{G}$.

Two seminal algorithm for enumerating all cliques are provided in [14]. The first algorithm is a basic recursive branch and backtrack technique. The second algorithm is an optimized variation that prunes the search space faster. The complexity of the second algorithm is $O\left(|V|3^{\frac{|V|}{3}}\right)$ [43]. Among all the variations [19, 26, 43, 44] of the algorithms in [14], [43] was able to reach a complexity of $O\left(3^{\frac{|V|}{3}}\right)$ by integrating the output function into the algorithm. This is optimal, since a graph can contain up to $3^{\frac{|V|}{3}}$ cliques [6] [37].

The maximal cliques enumeration algorithm in [43] has polynomial storage requirements, and is optimal in the worst case. However, since the sites' similarity graph is extremely sparse as shown in Section 6, quantifying the complexity in terms of the number of maximal cliques in the graph is crucial to our application. Although this involves complex analysis [43], the original experiments in [14] showed the average time to identify a maximal clique, i.e., a coalition attack, is not dependent on the size of the graph or the number of maximal cliques.

We recommend the implementation of [43] due to its optimal worst case complexity, though other algorithms established complexity bounds in terms of the number of maximal cliques in the graph, i.e., the number of coalitions. For instance, [44] combined the pruning techniques in [2] and [14] to find all the maximal independent sets with a complexity of $O(|V||E|\mu)$, where $|V|$, $|E|$, and μ are the numbers of nodes (sites), edges, and cliques in the graph. This algorithm was improved in [19] to $O(a(G)|E|\mu)$, where $a(G) \leq O(\sqrt{|V|})$.

6. FINDINGS ON A REAL NETWORK

We have devised *Similarity-Seeker* for detecting coalitions of pairs of attackers, and then extended it for coalitions of arbitrary sizes. To check the validity and the effectiveness of our development, we comment on our comprehensive set

[6]For instance, let G be a graph of x triplets, i.e., $|V| = 3x$. Two nodes are connected if they belong to different triplet. Hence, there are 3^x cliques, each of size 3.

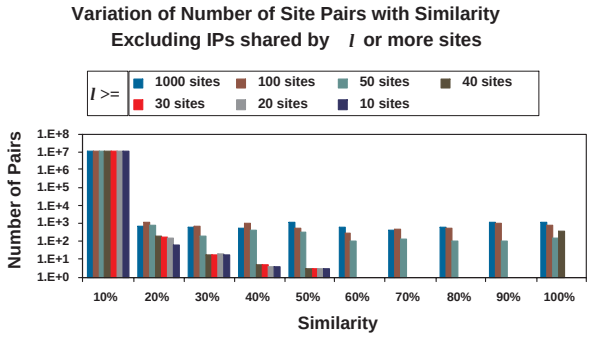


Figure 4: The Number of Pairs (Logarithmic Scale) Having a Specific Similarity.

of experiments using real data. We describe our experience with building a fraud detection system at Fastclick, Inc., a ValueClick company. For proof of concept, we analyzed a data sample of 54,045,873 traffic entries using *Similarity-Seeker* to discover site pairs with similar traffic. To reduce the noise, we excluded all the IPs that visited a large number of sites, because such IPs probably belong to NAT boxes. We repeated the experiment and progressively increased the parameter l, the number of sites beyond which the IPs are disregarded, from 10 to 1,000. The results are plotted in Figure 4, with a logarithmic scale on the vertical axis.

It is interesting to note that when l was 1,000, 98.94% of the pairs had less than 1% of similarity. When l was 10, 99.98% of the pairs had less than 1% of similarity. Clearly, the similarity between sites' IP sets is negligible. For each run, *Similarity-Seeker* output all the pairs with similarity more than 10%, and we fed the output to the cliques enumeration algorithm [43] to discover larger coalitions.

In particular, when l, the number of sites beyond which the IPs are disregarded, was 10, *Similarity-Seeker* output 81 pairs with similarity, s, at least 0.1. The 81 pairs contained 5 coalitions of size 3. As l increased, more pairs were discovered. For instance, when l was 30, 189 pairs of sites were found. Since more IPs shared by the disjoint components were considered, the cliques enumeration algorithm was able to connect some of these components into bigger coalitions. However, many of the cliques were overlapping.

As l increased to 40, 647 pairs were found. It became clear that the cliques discovered are highly overlapping and noisy, since popular IPs that are shared by many legitimate sites are not discarded. To reduce the noise, we increased the minimum traffic similarity gradually beyond 0.1. When s reached 0.5, for the same l of 40, the output of *Similarity-Seeker* comprised 406 pairs of sites that translated into exactly 1 perfect clique of size 29. That was clearly a *coalition* attack, as discussed below.

As l, the number of sites beyond which the IPs are disregarded, increases, more legitimate pairs are output as similar, since popular IPs shared by those sites are not discarded. To overcome this problem, it is advisable to increase the minimum required traffic similarity, s, as we increase l.

We increased l to 50 sites and increased s gradually up to 0.6. There were 680 pairs. With very few anomalies that can be manually identified, there was the original coalition of size 29, sharing 15 sites with another coalition of size 22. There were 8 other disjoint coalitions of sizes between 3 and 10, some isolated pairs, and a star of size 6.

Among all the discovered 680 sites, there were strong evidences that more than 93% of them were real fraudsters. Most of the coalitions had all their sites signed up with the commissioner around the same date. The noticeable characteristic of the coalitions of size 29 and 22 was that their traffic was of moderate size, yet coming from IPs all over the world. After further investigations, we found that the `Referer` fields in the HTTP requests were coming from pages that do not have the commissioner's advertisements; and sometimes no advertisements at all. We suspect the attackers had some form of readily available traffic through a network of Trojans, and that they do not work for the domains they signed up for. When the commissioner sent the account activation e-mails[7], the publishers somehow acquired the attached activation secrets and activated the accounts. Since the activation secrets are stored in a hashed form, the attackers must have compromised some machines on those domains, and hence, acquired the attached secrets.

For some of the isolated pairs, we noticed almost simultaneous traffic entries for the two sites of the coalition. We suspect that those attacks were launched using the attack in [3] that will be discussed shortly in Section 7.

As l grew further, the cliques enumeration algorithm did not connect those coalitions, but rather started to output groups that share extremely popular IPs. We checked those IPs on www.arin.net/whois, and they are ISP-owned IPs.

We recommend setting l and s initially to small values, say 5 sites and 0.1 similarity. From there, the commissioner can tune the values of l and s according to the noise in the results, as described above. The noise is usually manifested in the number of isolated pairs, small coalitions, and overlapping coalitions. From our experience, an appropriate value for the error ϵ is $\frac{s}{10}$, and the plausible range for the confidence α is between 0.01 to 0.1.

7. RELATED WORK

The work related to ours can be classified into two categories. The first category is the recent research in discovering densely connected subgraphs in huge graphs. The second category is our previous work on *coalition* attacks.

7.1 Discovering Dense Subgraphs

Due to the NP-hardness of the cliques enumeration problem, it was recently attempted to approximate this problem as discovering densely connected subgraphs or clusters.

[7]When a publisher signs up with FastClick, FastClick sends the publisher an e-mail on the domain signed up, with a secret key. The publisher can activate the account only using this secret key. This ensures that the publisher has an e-mail account on the domain (s)he signed up for.

Three main approximations were proposed. The *conductance* measure [8, 18, 22, 27] of a subgraph is a measure of the number of the external edges (bridges to the rest of the graph), in comparison to the internal edges of the subgraph, and the internal edges of the rest of the graph. The second *cluster editing* approximation [41] bounds the number of edges to be added or deleted to transform a subgraph into an isolated clique. The third approximation [1] bounds the average internal degree of nodes. However, discovering any cluster with a bound on any of the three approximations is proved to be NP-complete [23, 42]. Being as hard as the original problem, the approximations are of limited use.

The algorithm in [21] efficiently discovers dense clusters. However, the algorithm associates no connectivity metrics with the identified clusters. It hence provides no guarantees on the clustering quality. The algorithm cannot answer queries about clusters with a specific threshold on a connectivity metric, since the identified clusters can be either split or combined to satisfy the query threshold. Although the algorithm is suitable for *link spam* detection, it is not applicable in money-sensitive fraud detection.

7.2 Previous Work on Coalition Attacks

We have previously proposed an algorithm to detect the coalition attack identified in [3]. The attack in [3] involves a coalition of a dishonest publisher, P, with a dishonest Web site, S. S's page will have a script that runs on the surfer's machine when its page loads, and *automatically* redirects the surfer to P's Web site. P will have two versions of its Web page, a non-fraudulent page; and a fraudulent page. The non-fraudulent page is a normal page that displays the advertisement, and the surfer is totally free to click it or not. The fraudulent page has a script that runs on the surfer's machine when it loads, and *automatically* clicks the advertisement. P selectively shows the fraudulent page when the Web site that referred the surfer to P is S. The attack silently converts every innocent visit to S to a click on the advertisement in P's page. Several factors make the attack virtually impossible to detect. First, if the commissioner directly visits P's page, the non-fraudulent page will be loaded. Second, the commissioner cannot know the `Referer` fields of the HTTP requests to publishers. To identify S, the commissioner has to check all the Internet sites, which is infeasible. Third, the attack is done in an *automatic* way that is *hidden* from the surfer.

In [35], we proposed a solution this sophisticated *coalition* attack via a collaboration between commissioners and ISPs. By analyzing the aggregate stream of HTTP requests, the ISP can detect sites that are usually visited before a specific site, without violating the surfers' privacy. Bearing in mind the size and the speed of HTTP requests made to ISPs, the problem boils down to identifying associations between HTTP requests that are not widely separated in a traffic stream. We devised the *Streaming-Rules* algorithm to detect associations among stream elements.

Although the solution proposed in [35] is effective, it is very specific to the attack in [3]. The solution is not effective against other *coalition* attacks. For instance, if each attacker in the *coalition* controls a network of surfers' machines through Trojans, then HTTP requests for attackers could be widely separated in the ISP HTTP stream, and hence, are not detected by the solution in [35]. However, the general solution proposed in this paper detects *coalition* attacks in their full generality, including the attack in [3] without the need to the ISPs' support.

8. DISCUSSION AND FUTURE WORK

We have proposed a generalized solution for detecting generalized *coalition* attacks of *hit inflation*. Since sites' traffic is highly dissimilar, any similarity is usually suspicious. We modeled the problem of detecting fraud coalitions in terms of the set similarity problem. We built on several published theoretical results to propose 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 graph. On real network data, 93% of the detected sites were provably fraudsters. This shows how accurate our model and algorithm are regardless of how the attack is designed.

However, several publishers can collude to attack more than one commissioner. Each commissioner can only know the traffic of its publishers, and no commissioners can detect the similarity between the traffic of the attackers. Then, to detect attacks that span several advertising networks, we anticipate the development of new specialized auditing, or "detective", entities that are trusted by commissioners.

Our future work focuses on two directions. The first direction is to compare several cliques enumeration using real data. Although [43] has an optimal worst case bound, other algorithms can outperform it for extremely sparse graphs. The algorithms in [19, 20, 31, 43] were never experimentally compared, to the best of our knowledge. The second direction of our future work is to extend the algorithms to the streaming environment. Bearing in mind that an average-sized commissioner has around 50,000 publishers' sites, and receives around 70M traffic entries per hour, it is desirable to detect *coalition* attacks using only one scan on the data.

Acknowledgment

We thank Dr. Jerry Qi Zheng for helping us with acquiring the real data, and for his useful discussions.

9. REFERENCES

[1] J. Abello, M. Resende, and S. Sudarsky. Massive Quasi-Clique Detection. In *Proceedings of the 5th LATIN Latin American Symposium on Theoretical Informatics*, pages 598–612, 2002.

[2] E. Akkoyunlu. The Enumeration of Maximal Cliques of Large Graphs. *SIAM Journal on Computing*, 2(1):1–6, 1973.

[3] 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.

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

[5] 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.

[6] T. Bohman, C. Cooper, and A. Frieze. Min-Wise Independent Linear Permutations. *Electronic Journal of Combinatorics*, 7:R26, 2000.

[7] A. Bowker and G. Lieberman. *Engineering Statistics, 2nd Edition*. Prentice Hall, 1972.

[8] U. Brandes, M. Gaertler, and D. Wagner. Experiments on Graph Clustering Algorithms. In *Proceedings of the 11th ESA European Symposium on Algorithms*, pages 568–579, 2003.

[9] A. Broder. On the Resemblance and Containment of Documents. In *Proceedings of the IEEE SEQUENCES Compression and Complexity of Sequences*, pages 21–29, 1997.

[10] A. Broder. Identifying and Filtering Near-Duplicate Documents. In *Proceedings of the 11th COM Symposium on Combinatorial Pattern Matching*, pages 1–10, 2000.

[11] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-Wise Independent Permutations (Extended Abstract). In *Proceedings of the 30th ACM STOC Symposium on Theory Of Computing*, pages 327–336, 1998.

[12] A. Broder and U. Feige. Min-Wise versus Linear Independence (Extended Abstract). In *Proceedings of the 11th ACM-SIAM SODA Symposium on Discrete Algorithms*, pages 147–154, 2000.

[13] A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic clustering of the Web. In *Proceedings of the 6th WWW International Conference on World Wide Web*, pages 391–404, 1997.

[14] C. Bron and J. Kerbosch. Algorithm 457: Finding All Cliques of an Undirected Graph. *Communications of the ACM*, 16(9):575–577, 1973.

[15] 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.

[16] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th ACM STOC Symposium on Theory Of Computing*, pages 380–388, 2002.

[17] S. Chaudhuri, R. Motwani, and V. Narasayya. On Random Sampling over Joins. In *Proceedings of the 18th ACM SIGMOD International Conference on Management of Data*, pages 263–274, 1999.

[18] D. Cheng, S. Vempala, R. Kannan, and G. Wang. A Divide-and-Merge Methodology for Clustering. In *Proceedings of the 24th ACM PODS Symposium on Principles of Database Systems*, pages 196–205, 2005.

[19] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.

[20] L. Gerhards and W. Lindenberg. Clique Detection for Nondirected Graphs: Two New Algorithms. *Computing*, Volume 21(4):295–322, 1979.

[21] D. Gibson, R. Kumar, and A. Tomkins. Discovering Large Dense Subgraphs in Massive Graphs. In *Proceedings of the 31st VLDB International Conference on Very Large Data Bases*, pages 721–732, 2005.

[22] C. Gkantsidis, M. Mihail, and A. Saberi. Conductance and Congestion in Power Law Graphs. In *Proceedings of the 22nd ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 148–159, 2003.

[23] K. Holzapfel, S. Kosub, M. Maaß, and H. Täubig. The Complexity of Detecting Fixed-Density Clusters. In *Proceedings of the 5th CIAC Italian Conference on Algorithms and Complexity*, pages 201–212, 2003.

[24] P. Indyk. A small Approximately Min-Wise Independent Family of Hash Functions. In *Proceedings of the 10th ACM-SIAM SODA Symposium On Discrete Algorithms*, pages 454–456, 1999.

[25] 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.

[26] H. Johnston. Cliques of a Graph-Variations on the Bron-Kerbosch Algorithm. *International Journal of Computer and Information Sciences*, 5(3):209–238, 1976.

[27] R. Kannan, S. Vempala, and A. Veta. On Clusterings: Good, Bad and Spectral. In *Proceedings of the 41st IEEE FOCS Annual Symposium on Foundations of Computer Science*, pages 367–377, 2000.

[28] 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.

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

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

[31] E. Loukakis. A New Backtracking Algorithm for Generating the Family of Maximal Independent Sets of a Graph. *Computers & Mathematics with Applications*, 9(4):583–589, 1983.

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

[33] R. McGann. Study: Consumers Delete Cookies at Surprising Rate. *ClickZ News*, March 14 2005.

[34] 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.

[35] 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.

[36] A. Metwally, D. Agrawal, and A. El Abbadi. Hide and Seek: Detecting Hit Inflation Fraud in Streams of Web Advertising Networks. Technical Report 2006-06, University of California, Santa Barbara, Department of Computer Science, 2006.

[37] J. Moon and L. Moser. On cliques in graphs. *Israel journal of Mathematics*, 3:23–28, 1965.

[38] 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.

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

[40] 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.

[41] R. Shamir, R. Sharan, and D. Tsur. Cluster Graph Modification Problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

[42] J. Sima and S. Schaeffer. On the NP-Completeness of Some Graph Cluster Measures. In *Proceedings of the 32nd SOFSEM Conference on Current Trends in Theory and Practice of Informatics*, pages 530–537, 2006.

[43] E. Tomita, A. Tanaka, and H. Takahashi. The Worst-Case Time Complexity for Generating All Maximal Cliques. In *Proceedings of the 10th COCOON Annual International Conference on Computing and Combinatorics*, pages 161–170, 2004.

[44] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A New Algorithm for Generating All the Maximal Independent Sets. *SIAM Journal on Computing*, 6(3):505–517, 1977.

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

[46] J. Vitter. External Memory Algorithms and Data Structures: Dealing with Massive Data. *ACM Computing Surveys*, 33(2):209–271, 2001.

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