Hide and Seek: Detecting Hit Inflation Fraud in Streams of Web Advertising Networks *

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

Qi Zheng
FastClick, Inc.
360 Olive Street
Santa Barbara CA 93103
Jerry@fastclick.com

Abstract

As the Internet continues to grow, the Internet advertising industry flourishes as a means of reaching the appropriate market segments. Internet advertisers provide the monetary incentive for Internet publishers to display advertisements on their Web sites. Internet advertisers and publishers contract through a commissioner that takes care of the accounting issues, and earns a commission on the advertisers' payments. Some publishers are dishonest, and use automation to generate traffic to defraud the advertisers. The commissioner, who is supposed to distinguish fraudulent traffic from normal traffic, cannot track individual computers in order not to violate surfers' privacy.

In this paper, we describe the advertising network model in detail, and share, with the research community, our field experience and concerns about detecting fraudulent publishers. This paper provides a classification of the publishers' *hit inflation* techniques, and stream analysis schemes that would detect fraud attacks of many classes of the proposed classification. For some classes, we abstract detecting their fraud attacks as theoretical stream analysis problems that we propose as open problems. In addition, we consider the problem of detecting automated activities. A framework is outlined for deploying the proposed fraud detection algorithms on a generic architecture. We conclude by some findings of our attempt to detect fraud on a real network.

1 Introduction

As the Internet continues to grow, the Internet advertising industry flourishes as a means of reaching the appropriate market segments. Internet advertising is different from conventional TV/Radio advertising, which tends to be expensive, broadcast-based, and diluted. On the other hand, Internet advertisers are currently able to inexpensively direct their campaigns to the appropriate audience [5, 16, 33]. Current data management and Web programming technologies allow the classification of surfers (potential Internet customers), and thus allow the targeting of advertisements to the appropriate customer body on the fly. The main coordinators in this setting are the Internet advertising commissioners, who are positioned as the brokers between Internet publishers and Internet advertisers. In a standard setting, an advertiser provides the commissioner with its advertisements, and they agree on a commission for each customer action, e.g., clicking an advertisement, filling out a form, bidding in an auction, or making a purchase.

The publishers, motivated by the commission paid by the advertisers, contract with the commissioner to display advertisements on their Web sites. Since publishers are paid by the traffic they drive to advertisers,

*This work was supported in part by NSF under grants EIA 00-80134, NSF 02-09112, and CNF 04-23336.

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

there is an incentive for dishonest publishers to *inflate* the number of impressions and clicks their sites generate. In addition, dishonest advertisers tend to simulate clicks on the advertisements of their competitor to deplete their advertising budgets. This fraudulent behavior results in bad reputation for the commissioner, and sometimes in extra costs or paying reimbursements for advertisers [17].

One of the main challenges involved in detecting such fraudulent behavior is maintaining the privacy of the web users. This represents the standard conflict between security and the citizen's privacy. Furthermore, traditional approaches use off-line database techniques to detect fraud. This is unrealistic given the scale of the click stream in an ever growing Internet. In this paper, we depend on data analysis techniques that can be executed on an aggregate level such that the individuals' identities are not revealed, but still enabling the analysis algorithms to achieve satisfactory levels.

During the course of this paper, we will develop the first well-rounded exploration of the advertising network model. We adopt a compendious framework for identifying fraud in advertising networks. Instead of simply discovering an attack and building a detection algorithm, we classify most of the publishers' *hit inflation* fraud attacks into *non-coalition* and *coalition* attacks. The former class comprises single-publisher attacks that do not involve forming publishers-coalitions. On the other hand, fraud techniques that involve a coalition among publishers can be much subtler, and were studied in a limited number of works [1, 22]. For many classes of the proposed classification, we provide analysis schemes that detect fraud attacks of such classes. For the rest of the classes, we abstract detecting their fraud attacks as theoretical open problems. Finally, we discuss how those algorithms can be deployed on a generic Web server architecture [6] for the commissioners, and we conclude by some findings on a real network.

The rest of the paper is organized as follows. We start by describing the current state-of-the-art in advertising networks in Section 2. We follow by a review of previous work on discovering *hit inflation* fraud attacks in advertising networks in Section 3. In Section 4, we provide an overview and a classification of the fraud techniques employed for *hit inflation*. We propose a model for detecting fraud on a generic Web server architecture in Section 5. Our findings on a real network are reported in Section 6. We sum up the open problems and conclude in Section 7.

2 The Advertising Network Setting

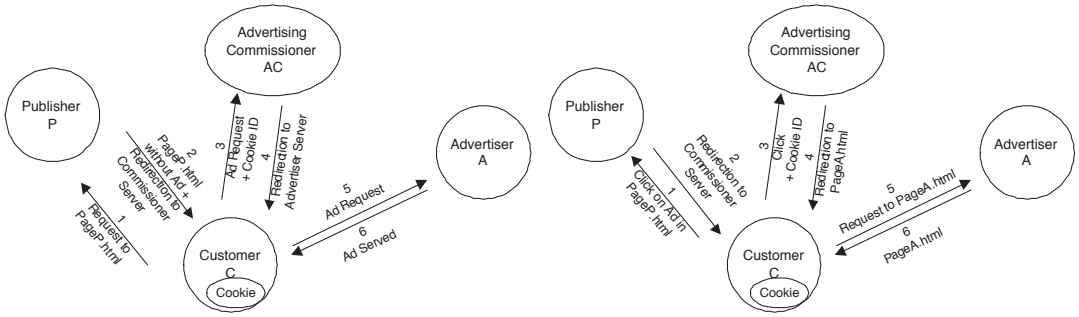
In this section, we discuss the advertising network model in detail. We start by describing the traffic models in Section 2.1, and discuss how the commissions are paid by the advertisers to the commissioners and publishers in Section 2.2. In Section 2.3, we identify the two main fraud activities in advertising networks.

2.1 The Traffic Models

The advertising commissioner contracts with both advertisers, who like to display their advertisements, and publishers, who are motivated by the commission. The commissioner keeps track of the advertisers' budgets so that they are not over-spent. In addition, it provides the publishers with the format of the HTTP requests that should be sent to the commissioner's servers for an impression (advertisement rendering), or a click. The commissioner receives the click and impression requests and processes them as discussed below.

The impression traffic models the steps between a surfer requesting the publisher's page, and the advertisements being loaded along with the requested page on the surfers' Browsers, as is depicted in Figure 1(a). When a surfer requests the page (step 1), the publisher loads its page on the surfer's Browser (step 2), and redirects the Browser to the commissioner's server (step 3). Since the commissioner stores the recent advertisements shown to the surfer in his/her cookies, it avoids showing repetitive advertisements[1]. Among

[1]This way the commissioner also gives the advertisers better exposure. This process is called *frequency-capping* [24].



(a) Impression Traffic in Advertising Networks.

(b) Clicks Traffic in Advertising Networks.

Figure 1: The Traffic Models in Advertising Networks.

the advertisements not shown recently for that surfer, the commissioner picks the advertisement that maximizes the commissioner's returns (*effective-Cost-Per-thousand-iMpression* or (*eCMP*)) to be displayed, and logs the impression for accounting purposes. The commissioner, then, redirects the surfer's Browser to the advertiser's server (steps 4 and 5) that loads its advertisement on the Browser (step 6)[2].

The click traffic is similar to the impression traffic. Whenever a customer clicks a link on the publisher's Web page, the customer is referred to the server of the advertising commissioner, who logs the click, for accounting purposes, and *clicks-through* the customer to the Web site of the advertiser, who loads its page on the Browser. The click traffic is illustrated in Figure 1(b).

2.2 Paying Commissions

The publishers are paid based on the impression and click traffics they drive to the advertisers' Web sites. The commissioner earns a percentage of this revenue. The commissioner's percentage is fixed if the advertiser pays using the same scheme as the publisher. For instance, if the advertiser pays according to the number of impressions (clicks), and the publisher is paid by impressions (clicks) too, then the commissioner earns a fixed percentage of the payment.

However, if the advertiser pays according to the number of sales, and the publisher is paid by impressions or clicks, then this is called an *arbitrage campaign* [31]. In general, *arbitrage campaigns* are campaigns where the commissioner is being paid by the advertiser according to a metric different from the one it uses to pay publishers. In the case of *arbitrage campaigns*, the commissioner has to make sure that it earns from the advertiser more than what it pays to the publisher. Otherwise, the commissioner will be making a loss. In the real world, the majority of the Internet advertising campaigns are *arbitrage campaigns*, since the advertiser prefers to pay according to its sales, while the publisher favors being paid according to its servers' load, which corresponds to the number of impressions and clicks[3]. Nevertheless, some advertisers prefer to pay by impressions for product branding [19].

2.3 Fraud in Advertising Networks

In general, there are two main kinds of fraud in advertising networks, *hit shaving* [28], and *hit inflation* [1, 3, 12, 21, 22, 35]. *Hit shaving* is the fraud performed by an advertiser, who does not pay commission on some

[2]Sometimes, the commissioner stores the advertisements on its server to avoid steps 5 and 6.

[3]If all publishers are paid by the number of sales made through their traffic, then the advertising network is called an *affiliate program* or *CPA networks* [34]. It is beyond the scope of the paper to discuss fraud in this setting.

of the traffic received from publishers, by claiming that it received less traffic than the true numbers. Several schemes were proposed in [28] so that the publishers can estimate upper and lower bounds on the number of clicks their sites drive to the advertisers' sites. The number of clicks received by the commissioners is only an upper bound on the traffic driven from publishers to advertisers. However, the requests to the advertisers sites can be lost due to network problems. In [28], a design for publishers' sites was proposed so that the publisher and/or the commissioner can be notified once the advertiser's site is loaded on the Browser of the surfer. The focus of this work is on *hit inflation*, since it is a critical problem, and the problem of *hit shaving* has been well studied by Reiter *et al.* in [28]. Throughout this work, we concentrate on fraud detection, and thus, assume that the source IP addresses of the traffic are not spoofed. Spoofing is an orthogonal problem that is being studied in the network security community [7]. In addition, we assume the internal company policies and access restrictions deter insider attacks from employees.

Hit Inflation Attacks

Since publishers are paid by the traffic they drive to advertisers, there is an incentive for them to falsely increase the numbers of impressions and clicks their sites generate [35]. This phenomenon is referred to as *hit inflation* [1]. The simplest way to perform *hit inflation* is to employ a script that would continuously load the publisher's page and simulate clicks on the displayed advertisements [21]. In general, forged impressions are those that do not yield valuable exposures of the advertisements to the customer body; and forged clicks are those that do not reflect the interest of customers in the advertised products. In other words, forged traffic yield to a sale with probability $= 0$.

Fraud is not solely practiced by publishers. Advertisers are motivated to conduct fraud too, albeit for different reasons. Some advertisers perform *hit inflation* attacks on competitor advertisers. Such advertisers employ techniques that would continuously visit Web sites where they expect to find their competitors' advertisements and click on the displayed advertisement if it belongs to one of their competitors. Although this does not directly generate revenue to the fraudulent advertiser, it depletes the advertising budgets of its competitors, and hence limits the exposure of its competitors to the market. Advertisers' *hit inflation* is rarely found in advertising networks, since the advertisements change on the same page for the same surfer[4]. Hence, it is difficult to find the advertisements of competitors in advertising networks. Advertisers' fraud is more prevalent on search engines, which is another form of advertising on the Internet. Search engines' advertising programs are usually pay-per-click, and display roughly the same advertisements for consecutive loads of the same page, which is known as *static placement* of advertisements.

In addition to advertisers' fraud, some scripts, or *robots* [23] such as *spiders* employed to crawl the Web by search engines, generate traffic to sites without the publisher having a malicious intention. However, the traffic of such legitimate *robots* have well-known agent signatures, are easily identified by commissioners, and are disregarded. The major sources of *hit inflation* are publishers and advertisers. In this work, we focus primarily on publishers' fraud, since the discussion can be generalized to advertisers' fraud.

One of the rôles of the advertising commissioners is to identify automated activities. First, this improves the reputation of the commissioner, and attracts more advertisers, since they only pay for valuable impressions and clicks. Second, this stops potential losses of the commissioner on *arbitrage campaigns*, since the commissioner does not pay publishers for automated fruitless traffic.

3 Proposed Solutions for Hit Inflation

In this section, we review the work on detecting *hit inflation* in advertising networks. The inherent problem of falsely increasing the number of impressions and clicks, as described in Section 2.3, has been a

[4]This is called *dynamic advertisements placement* [26].

concern to the commissioners since their conception, especially since real life fraud techniques are rarely published [1, 15]. We give examples of classical fraud detection measures in Section 3.1. We discuss the cryptography-based approaches in Section 3.2, and observe that the commissioner cannot track individual personal computers to identify fraud, since this violates the privacy of the surfers. Then, we argue that the legitimate way to identify fraud is to perform statistical analysis on aggregate traffic streams in Section 3.3.

3.1 Classical Fraud Detection

Classical (off-line) fraud detection requires the commissioners to store the entire traffic in databases and to check the quality of the stored traffic through complex SQL scripts. Below, we give some examples.

The most effective off-line metrics is the *Click Through Rate* (*CTR*)[5]. *CTR*s that deviate significantly from the rest of the network, on the same advertisements, are suspicious, since the *CTR* of an advertisement is roughly consistent across sites of the same nature. This test is based on an observation in [15]. By the same token, publishers who have consistent *CTR*s across all advertisements are suspicious.

The second off-line metric is motivated by the *Cost-Per-Action* (*CPA*) campaigns where publishers pay according to the number of sales or actions made. Most of the *CPA* campaigns on advertising networks are *arbitrage campaigns*, where the publisher is paid according to the traffic volume, but the advertiser pays the commissioner according to the sales volume. The *CPA-Score* of a publisher is the profit margin of the commissioner on the traffic of this publisher. A *CPA-Score* of 0 means that the publisher incurred as much cost to the commissioner as it generated revenue. Since commissioners have a good percentage *arbitrage campaigns*, and use *dynamic advertisements placement*, it is fair to compare publishers using this measure.

The *Rich Media Click Rate* (*RMCR*) of a publisher is the number of its clicks on animated (rich-media) advertisements as a percentage of clicks on both animated and static (image) advertisements. There is a stable network-wide ratio between clicks on animated and clicks on image advertisements. It is technically more difficult to simulate clicks on animated advertisements, since the script has to parse the advertisement, this ratio gets violated if the publisher forges its clicks.

There are two main problems with the classical off-line measures. First, off-line measures lack scalability. Keeping in mind that, currently, an average size commissioner receives 20K impressions per second, which correspond to 70M records added to the database every hour, and these numbers continue to grow, it is no wonder that running scripts to calculate those metrics degrade the performance of the database to the limit that commissioners run them only occasionally. Updating those metrics is not scalable, too, since it is clear that one click on an advertisement at any site might entail various recalculations of statistical measures and re-ranking of sites.

Second, and more important, parameterizing the off-line measures guarantees some quality of the traffic accounted for, but cannot be used as an absolute basis for discovering fraudulent traffic. Those tests check only for the quality of the traffic, and not the malicious intention of the publisher.

For instance, some non-professional sites have poor content, which affects the sales rate (*Conversion Rate* or *CR*), or the *CTR*. Other sites, e.g. kids and games sites, have a very consistent high *CTR* across all advertisements. The reason is that a lot of the games' clicks, by mistake, go to the advertisements. Even more, some sites request only non-rich-media advertisements, since they are faster to load on the surfers' Browsers. This deviates the *RMCR* of such sites from the rest of the network.

Aggressively discarding low-quality traffic conserves the advertisers' budgets, though it is offensive for both publishers and commissioners who are not paid for all the traffic their servers delivered. Conversely, setting tolerant parameters results in the advertisers paying for fraudulent traffic. Since only the fraudulent entities can identify their fraudulent traffic, and there are no specific parameter values that discard only and nothing but the fraudulent traffic, this leaves commissioners in a dilemma. Commissioners cannot use those

[5]The *CTR* of a publisher, advertiser, or advertisement is the number of clicks it receives as a percentage of impressions.

metrics to exactly filter out fraudulent traffic, and have to choose between discarding some legitimate traffic, and hence sacrificing some revenue, and charging the advertisements for traffic that is not all legitimate, and hence risking being sued by advertisers [17].

3.2 Cryptographic Approaches Versus User Privacy

Several cryptographic approaches were proposed to substitute the off-line measures. Their central idea is change the industry model so that dishonest publishers have less chance to commit fraud. In [12], Jakobsson *et al*. proposed a simple scheme using e-coupons. The advertiser would generate some cryptographic e-coupons and distribute them to the publishers. The publishers would then redistribute the e-coupons to the surfers. The surfers would use the e-coupons for purchases from the advertisers.

The scheme implicitly encourages pay-per-sale that is in favor of advertisers. Many publishers were reluctant to be paid except according to impressions and clicks since this corresponds to the load on their servers. Advertisers can make use of the model to receive a lot of impressions and clicks, which are important for branding, while giving low-value e-coupons. In addition, this model is more prone to *hit shaving* by advertisers.

Other cryptographic approaches require the cooperation of the surfers to distinguish fraudulent traffic from normal traffic. In [3, 25], protocols were proposed to count the number of visits to a Web site using cryptographic primitives. The framework requires surfers to initially register with the commissioner. During this registration, the commissioner sends the surfer a token for free services redeemable at the publisher's site; and sends the publisher a corresponding token so that the publisher can identify the registered surfers. Whenever the surfer visits the publisher, (s)he can access the free service if (s)he sends the authentication token to the publisher. Every time the authentication token is sent to the publisher, the surfer updates it using a hash function. The publisher can verify the token value, but cannot anticipate the value of the next visit. At accounting time, the number of visits of a surfer is documented by the last token value sent to the publisher.

This scheme has some problems. First, it assumes that the surfers trust commissioners to download code [13] that can perform the hashing and the communication with the publishers' servers. Second, it requires a large number of hash functions, ideally, one for each surfer. Third, and most important, the commissioner has to uniquely identifies surfers, which violates their privacy. This can be done either by the surfers registering with the commissioner, and revealing some unique personal information, or by the commissioner downloading spyware [30] on customers' machines to track them by the serial numbers of their hardware or software installations.

3.3 Detecting Fraud Through Data Management

We propose using data analysis techniques for the purpose of fraud detection based on a fourfold reasoning. First, they can be executed on an aggregate level such that the individuals' identities are not revealed, but still enabling the analysis algorithms to achieve satisfactory levels. Although advertising commissioners would like to track each machine by a unique ID, and know how many users are using the machine for both advertisement targeting [29], and fraud detection purposes [21], this compromises users' privacy. Advertising commissioners track individual customers by assigning distinct customer IDs in cookies set in the customers' Browsers [24]. The fact that customers are totally free to accept, reject, or periodically clear such cookies, preserves their privacy [20]. Hence, commissioners currently rely on two legitimate pieces of information to identify computers, the IP address, and the cookie ID.

Second, we argue that data management tools can help identifying prevalent patterns and correlations that exist in abnormal traffic exclusively. Hence, data analysis techniques can evidence malicious intentions

in order to complement the classical off-line measures that only detected low-quality traffic and was unable to distinguish it from traffic generated with malicious intentions.

Third, data management tools can be used to reduce the database load, by limiting the complex SQL analysis to the publishers and advertisers that receive suspicious traffic. Once a suspicious pattern is detected in the traffic of a publisher or an advertiser, it is easy to look for off-line poor performance indicators, rather than analyzing the traffic of each publisher and advertiser.

Fourth, detecting fraud through data analysis does not entail changing the business model. We advocate relying on the the IP addresses, and the cookie IDs for identifying suspicious traffic characteristics, and hence data analysis is expected to be widely accepted in the industry. The challenge now is that because dishonest publishers are aware of the limitations in tracking customers' machines, it is possible to conduct sophisticated frauds that frequently change the identification of the computers generating the fraudulent traffic, so that they are more difficult for the commissioners to detect. Thus, we have translated the fraud detection problem into a "hide and seek" game.

Examples from Data Management

Distinguishing fraudulent traffic from normal traffic, especially that real life fraud techniques are rarely published [15]. Due to the problem scale, fraud detection in advertising networks is a quintessential streaming application. The data received by the commissioners is massive to the limit that its storage cost outweighs its benefit. This data needs to be analyzed, aggregated on different dimensions, and searched for dubious patterns.

Recently, we have proposed methods for detecting *hit inflation* in traffic streams [21, 22]. The basic premise is that the advertising commissioner should be able to tell whether the impressions and clicks generated at the publisher's site are authentic, or are generated by a script running on some machines. As discussed in Section 2.3, a naïve way to defraud is to run a script that continuously loads the publisher's page and simulates clicks on the advertisements in the page. However, this leads to duplicate impressions and clicks within a short period of time, an hour for example. In [21], we developed a solution, based on Bloom Filters [2], to effectively identify duplicates in traffic streams to detect such fraud.

The solution employs a bit vector, *Hash-Vector*, and d independent hash functions, $h_1, h_2, \ldots, h_d$. The data structure and the algorithm are sketched in Figures 2, and 3, respectively. Assuming duplicate clicks are being checked, every time a click is observed, the click identification (IP address or cookie ID, advertisement ID, and publisher ID) is hashed using the independent hash functions, and the resulting indexes are checked in the bit vector. If at least one bit is 0, then this click is not a duplicate. Otherwise, the click is most probably a duplicate. After checking the click for duplication, the click is inserted into the bit vector by turning to 1 all the bits corresponding to outcome of the hash functions.

The errors of this scheme are only false positives. That is, it can only err by judging a click to be a duplicate, while the click is not really a duplicate. However, the scheme does not miss any real duplicates. The work in [21] provides probabilistic analysis for the error rate, and the space usage. It was found that the expected number of erroneous duplicates found in a stream of size N is $\sum_{i=1}^{N}(1 - e^{-d*(i-1)/M})^d$, which is less than $N * (1 - e^{-d*N/M})^d$, where d is the number of the hash functions, and M is the size of the bit vector. In general, the number of the hash functions is a measure of the processing time per click, and the number of bits used is roughly the consumed space.

A more sophisticated known *hit inflation* attack was identified by Anupam *et al.* in [1]. The attack is sketched in Figure 4. It 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 customer's machine when its page loads, and *automatically* redirects the customer 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 customer

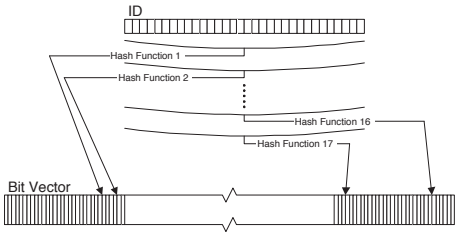


Figure 2: The Bloom Filters Based Solution for Duplicate Detection in Click Streams, $d = 17$.

Algorithm: ***Duplicate-Detector*****(Vector** ***Hash-Vector*****)**
begin
for each entry, e, in the stream{
Boolean duplicate = **True**;
for (Integer $i = 1, i \leq d, i++$){
Integer $hash = f_i(e)$;
if (*Hash-Vector*$[hash] = 0$){
//e cannot be a duplicate
duplicate = **False**;
}
Hash-Vector$[hash] = 1$;
}// end for
if (duplicate = **True**){output e as a duplicate;}
}// end for
end;

Figure 3: The *Duplicate-Detector* Algorithm.

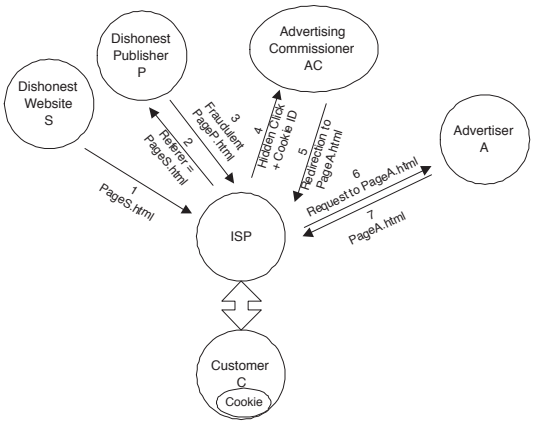


Figure 4: The Hit Inflation Attack in [1].

is totally free to click it or not. The fraudulent page has a script that runs on the customer's machine when it loads, and *automatically* clicks the advertisement. P selectively shows the fraudulent page when the Web site that referred the customer to P is S. P can know this information through the `Referer` field of the HTTP request. The attack will silently convert 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 try all the sites on the Internet, which is infeasible. Third, the customers' IDs visiting P's page will be distinct. Hence, the commissioner cannot detect this fraud using a duplicate detection technique. Fourth, the attack is done in a way that is *automatic* and *hidden* from the customer.

In [22], a solution was proposed for detecting this sophisticated *coalition* attack via a collaboration between commissioners and Internet Service Providers (ISPs). By analyzing the aggregate stream of HTTP requests, the ISP can detect which sites are usually visited before a specific site, without violating the customers' privacy. Bearing in mind the size and the speed of HTTP requests made to the ISP, the problem boils down to identifying associations between elements in a stream of HTTP requests. In [22], the *Streaming-Rules* algorithm was devised to detect associations in a stream of elements.

4 An Overview of Fraud Techniques and Detection Algorithms

We start by providing a classification of the most prevailing *non-coalition*, and a discussion of *coalition* attacks, in Sections 4.1, and 4.2, respectively. Then, we discuss some techniques to detect robotic behavior in Section 4.3. Through the sequel, we concentrate on fraud detection, and thus, assume that the source IP addresses of the traffic are not spoofed. Spoofing is an area of research that is being studied in the network security community [7]. In addition, we assume the internal company policies and access restrictions deter insider attacks from employees.

4.1 A Classification of Non-Coalition Publishers' Attacks

We base our classification on the way the commissioners identify surfers' machines, i.e., on the basis of the IP address and the cookie ID of the machine. The normal surfers' traffic that visits a publisher's site has stable characteristics that are violated if the traffic is automated. Normally, the traffic encodes some relationship between IP addresses and cookie IDs. The imbalance in the relationship between the IP addresses and the cookie IDs is what we, as fraud detectives, look for in order to select a subset of the publishers for further manual investigation.

The attack technique that a fraudulent publisher uses has to leave a fingerprint on the relationship between the IP addresses and cookie IDs of the traffic. The attack is either coming from one or several IP address. The number of cookies used on the machines used in the attack can be none, one or more. There are six possible combinations of IP addresses and cookie IDs used in the attacks. Table 1 summarizes the combinations, and the fingerprint to look for in order to discover each class of attacks. It will become clear in the sequel that the more cookies and IP addresses the attacking publisher controls, the more difficult it is to detect its attacks.

4.1.1 Cookie-Less Attacks

In case the publisher disables the cookies on the machines used for the attacks, then this situation fits into the first column of the table. Whether the publisher is directing its attack from one or more IP addresses, this publisher will have a high percentage of its traffic being cookie-less, and this can be easily detected by

IP-Cookie Relationship		Cookie ID		
		0	1	Many
IP	1	High Percentage of Cookie-less Traffic	Duplicates and u-Duplicates	Unbalanced activities of IPs and Publishers
	Many		A cookie occurring in many IPs	Random: A cookie occurring in many IPs
				Preassigned: Unbalanced activities of IPs and Publishers

Table 1: A Classification of the *Non-Coalition* Publishers' *Hit Inflation* Attacks.

monitoring the percentage of cookie-less traffic for each publisher and investigating those publishers that deviate considerably from the norm.

4.1.2 Single Cookie and Single IP Address Attacks

An attacking publisher can run a simple script, with no cookie manipulation, on one machine connected to the Internet through a fixed IP address. An example is given in [15]. The script visits the site multiple times to take advantage of the *dynamic advertisements placement*. Since distinct advertisements are displayed for consecutive visits to the same page, the definitions of duplicate impressions and clicks need to be revisited. For example, the commissioner cannot define two consecutive clicks, with the same IP address and cookie ID, as duplicates if they are on different advertisements on the same page, since it could be a real surfer who is interested in both advertisements. The commissioner needs to set a limit on the number of acceptable impressions and clicks from one pair of an IP address and a cookie ID on the same site per time period. The duplicate detection technique in [21] should be modified to report a click or an impression on a site only after being repeated several times, u, within a specific period, t.

We propose a preliminary solution here, which is an extension of our Bloom based solution in [21]. The algorithm employs a vector of integers, instead of bits, as well as d independent hash functions. To report a pair of an IP address and a cookie ID that has made more than u clicks, each integer should be capable of counting up to $u + 1$. Initially all the integers are initialized to 0. Every time a click is observed, it is hashed using the independent hash functions, and the integers corresponding to the results of the hash functions are checked. The algorithm should alert on the last click only if all the integers have values equal to $u + 1$. Otherwise, the click has not yet occurred u times. After checking the click, it is inserted into the integer-vector by incrementing all the integers that correspond to the results of the hash functions. The u*-Duplicate-Detector* algorithm is ommitted for space constraints. It is similar to the *Duplicate-Detector* algorithm in Figure 3, except that the *Hash-Vector*$[hash] = 1$ statement is replaced with *Hash-Vector*$[hash] + +$; and the test condition is changed from **if** (*Hash-Vector*$[hash] = 0$) to **if** (*Hash-Vector*$[hash] < n$). The error analysis of this scheme is similar to that developed in [21] and summarized in Section 3.

4.1.3 Single Cookie and Multiple IP Addresses Attacks

This class of attacks is typically more prevalent with dishonest advertisers, than with dishonest publishers. As described in Section 2.1, for any surfer, the commissioner displays the most profitable advertisements that were not shown in the recent history. Sending the same cookie repeatedly results in displaying the roughly same advertisements. The attacking advertiser can make several visits to a publisher's site until the advertisements of its competitors are displayed. Then, the attacker saves the cookie, and repeatedly uses it to make the commissioner display the advertisements of his competitors, simulate clicks on them, and deplete his competitors' budgets. We discuss our experience with this attack in Section 6.

Since it is easier for an attacking publisher to change the cookie than to change the IP address of the attacking machines, this kind of attack is not widespread among publishers. However, we discuss it here, since it is crucial for discussing attacks with multiple cookies and multiple IP addresses in Section 4.1.5.

Formally, we need to discover cookies that appear with more than p IP addresses in a specified time period, t. Since both the population of the IP addresses and the cookies are huge, from a stream management perspective, it is challenging to identify cookies that appear with an excessive number of IP addresses. This is an open research problem. However from an pragmatic point of view, this problem can be solved.

The solution stores the IP addresses on the cookies with the timestamps of the impressions. Every time an impression request arrives, the commissioner replaces the cookie reported with a new cookie. The new cookie is the same as the old one, after appending the IP address and the timestamp of the request. Hence, the commissioner can employ an algorithm to check the number of IP addresses associated with the cookie making the request, and alerts if there are more IP addresses than p in the last specified time period, t.

The problem with this solution is that the commissioner has to trust the data stored in the cookies. Bearing in mind that the cookies are stored on machines that could be performing the fraud, this assumption is unrealistic. One way around this problem is to store a hash-based checksum on the cookie that would be violated if the cookie was modified to delete some of the IP addresses. However, this solution does not safeguard against *cookie-replay* attacks [14].

A *cookie-replay* is an attack where the same unmodified cookie is sent to the server several times. In our context, the attacking publisher stores a copy, *old-cookie*, of the cookie, before sending it to the commissioner. When the commissioner appends the IP address to the cookie and sends it back to be stored on the machine, the attacking publisher replaces the modified copy by *old-cookie*. Next time, the attacking publisher sends *old-cookie* again to the commissioner and the same scenario is repeated. Effectively, the attacking publisher gets around the commissioner's technique of appending IP addresses to the cookie.

The *cookie-replay* attack was discussed in [14] in the context of identity theft. A cookie, carrying a user name and a password, sent from a user to a server for authentication, can be intercepted by a third party that eventually sends the same cookie to the server to steal the identity of the user. Several solutions for the *intercept-replay* scenario, some of which need some modification to the current Browser technology, were provided in [14]. The proposed solutions prevent eavesdropping, and thus, do not apply in our case since the cookies are stored on the attacking machines, and hence the attacking publisher does not need to perform any eavesdropping.

We now provide a solution for the *cookie-replay* attacks in advertising networks, that can also be applied in other contexts. The solution lies in storing a counter on the cookie. The counter is incremented every time the commissioner receives the cookie, and altering the counter will violate the checksum stored on the cookie. A duplicate detection technique is employed to detect duplicates in the stream of cookies making the requests. However, the d hash functions of the duplicate detection mechanism are seeded with the counter of the hashed cookie. Thus, if and only if the attacking publisher sends the same cookie back with the same counter value twice, it will be detected as a duplicate and the impression or click will be identified as fraudulent. The complete framework is sketched in Figure 5.

For example, if a cookie has made 5 legitimate impressions, it will have a counter with value 5. On the sixth impression, the algorithm will insert the cookie ID into the bit vector with the hash functions seeded with 5; increment the counter to 6; append the last IP from which the cookie appeared; and store the cookie on the surfer's machine. If the attacking publisher sends back the same old cookie for a seventh impression with a counter value of 5, the hashes will collide with the 1 bits resulting from inserting the cookie ID when the counter was 5. If the counter is modified or some IP addresses were deleted, the checksum gets violated.

```
Algorithm: p-IPs-Detector(Vector Hash-Vector)
begin
  for each cookie, c, in the stream{
    //Checking for cookie modification
    if (c.Checksum is violated){output c as modified;}
    //Checking for cookie replay attack
    Boolean replay = True;
    for (Integer i = 1, i ≤ d, i + +){
      Integer hash = f_i(c.Counter, c.ID);
      if (Hash-Vector[hash] = 0){
        //c cannot be a replay
        replay = False;
      }
      Hash-Vector[hash] = 1;
    }// end for
    if (replay = True){output c as a replay; Continue;}
    //Check if more than p IPs were observed in t
    delete from c IPs older than t;
    if (|c.DistinctIPs| > p){output c as exceeding p; Continue;}
    //Increment the counter of the cookie
    c.Counter + +;
    //Append the observed IP to the cookie
    append (ObservedIP, CurrentTime) to c.IPs;
    //Update the checksum
    CalculateChecksum(c);
    //Store back the cookie
    SendCookie(c);
  }// end for
end;
```

Figure 5: The *p-IPs-Detector* Algorithm.

4.1.4 Multiple Cookies and Single IP Address Attacks

This kind of attacks could be performed in several ways. The most straightforward, though not the most economic, way is to employ several scripts running on several machines connected to the Internet through a single router. Hence, several requests will be coming from the same IP address with several cookie IDs. The problem with this attack is that it resembles normal Internet traffic, where several users with different cookie IDs connect to the Internet using one IP address via a Network Address Translation (NAT) box or an ISP. A more sophisticated version of this attack is possible by connecting to the Internet through an ISP, so that the same IP address used in the attack is also shared with legitimate traffic. By mixing the fraudulent traffic with the normal traffic, the attacking publisher aims at diluting the effect of the attack, and confusing the commissioner's fraud detection mechanisms.

Detecting such attacks is an open problem. The commissioner needs to detect imbalances in the activities of IP addresses and publishers' sites, i.e., anomalies that do not reflect the average activity of the IP address and the site. Even formalizing the problem is a challenge. One way is fitting a 2-dimensional distribution for the traffic, and picking pairs that exceed the expectation with some threshold. Another way of formalizing the problem is to normalize the traffic of the publishers' sites, normalize the traffic of the IP addresses, and look for pairs that do not agree with the product of the normalized traffic distributions. The main challenge is the large cardinality of both IP addresses and sites, and the authors are unaware of any technique that can be readily employed to solve this problem in a streaming environment.

4.1.5 Multiple Cookies and Multiple IP Addresses Attacks

This is the class of attacks that is most difficult both to perform and to detect. The attacking publisher must have access to several valid cookies, and several IP addresses. There are several ways the attacking publisher can use several IP addresses and cookies. We will mention a few here. The most straightforward, though not the most economic, way is having several machines with several accounts on ISPs at the disposal of the attacking publisher. Another way is exploiting the cookies and IP addresses of legitimate users through spywares and Trojans [32], that would simulate impressions and clicks for the attacker's site by issuing the appropriate HTTP requests. Network anonymization could be used, as well. Those services are commercially available through www.anonymizer.com, www.silentsurf.com, and others. Most of those services block third party cookies. If so, the attack will be viewed as a cookie-less attack. The traffic generated by this attack resembles real traffic to a high extent.

We can view this class as a more complex case of some of the aforementioned classes. Assume the publisher has access to several valid cookies, and IP addresses, every time an impressions/click is made with one of the cookies, it is either preassigned to a specific IP address, or an IP addresses is picked randomly.

If every time a cookie is used in the attack, a preassigned IP address is used, then this kind of attacks can be viewed as a more complex instance of the "Multiple Cookies and Single IP Address" case, but using several IP addresses. On the other hand, if the IP address used with the cookie is randomly selected, then this will lead to having the same cookie appearing with several IP addresses. This can be viewed as a more complex instance of the "Single Cookie and Multiple IP Addresses" case, with several cookies.

Although we formulated this class in terms of simpler classes, this is not an ultimate solution. Since the attacking publisher has access to several cookies and IP addresses, it can perform the attack such that it just stays below the statistically acceptable thresholds (radars). For instance, if the attacker picks a cookie ID and an IP address at random and makes a request, it can still make plenty of requests without being detected. If the commissioner checks for cookies that appear with more than p IP addresses in a specified time period, t, then in one time slot of length t, the attacking publisher can make up to $p - 1$ requests for every valid cookie it possesses, given it has control over at least $p - 1$ IP addresses. A similar argument can be applied for the case where this attack is viewed as a scale-up of the "Multiple Cookies and Single IP Address" class. Reducing the thresholds of the "Multiple Cookies and Single IP Address" and "Single Cookies and Multiple IP Address" will not help much, since this will lead to a lot of false positives, i.e., legitimate publishers being reported as suspects. Hence, detecting this class of fraud remains an open problem.

4.2 Publishers' Coalition Attacks

Section 4.1 emphasizes the understandable correlation between the complexity of launching an attack, and the complexity of detecting it. The attacks whose detection mechanisms are complex, and left as open problems are those that involves personifying multiple identities. Disguising as many users is either costly, since the attacking publisher should have several proprietary connections to the Internet; or unscalable, since the attacker should control several surfers' computers through Trojans, for example. Therefore, publishers are motivated to form coalitions to reduce the cost and the difficulty of launching sophisticated attacks.

For instance, assume the commissioner pays publishers for up to u impressions per t time units for each pair of IP address and cookie. Attacker A can generate u hits from each identity (resource) it controls, without being rejected by the commissioners' radars. Another attacker, B, can simulate another u undetectable hits per controlled resource. Hence, it is more scalable and cost effective for both A and B to time-share the resources, and generate $2u$ hits for each publisher, rather than doubling the amount of controlled resources.

One way attacking publishers can share resources is the smart attack identified in [1], and summarized in Section 3.3. Publisher A can exploit the IP addresses and the cookies identifications of the surfers visiting its site. Publisher A can load the Web page of B in a frame of size 0, when loading its own Web page on

surfers' Browsers. Thus, each visit to A's site will generate one hit for A and another for B. B can follow the same example to pay A back. The same logic applies for other resources, whether they are spyware-based or owned by the publishers. In order to stay under the radars' levels, both A and B use a wider range of resources while generating u visits from each resource every t time units.

The phenomenon the commissioner has to look for, to detect *coalition* attacks, is the high similarity of the traffics of sites. For each publisher, A, given the set S_A of IP addresses (or cookies) that visit A, the commissioner needs to look for pairs, (S_A, S_B), of publishers with high similarity of their sets. A straightforward measure of similarity between S_A and S_B is the Jaccard coefficient, calculated as $\frac{S_A \cap S_B}{S_A \cup S_B}$. Normally, for any two independent sites, the Jaccard coefficient based on Cookie IDs is negligible. The Jaccard coefficient based on IP addresses should be slightly higher due to the NAT boxes.

Bearing in mind that an average-sized commissioner has around 50,000 publishers' sites, and that any naïve solution entails testing all pairs of publishers, the problem is intractable in real time. Fortunately, the algorithm proposed in [4], to determine the similarity of files on the Internet, can be employed here. The *Shingling* algorithm [4] calculates a sketch, a light-weight sample of terms, for each document. Then, it calculates the Jaccard coefficient of two documents based on the similarity of their sketches. To avoid the quadratic comparison of each possible pair of documents, the algorithm checks only the pairs whose sketches overlap, i.e., the pairs that have common terms in their samples. Hence, the complexity drops down to the number of terms multiplied by number of the pairs sharing a term. The algorithm is applicable in our case, where sketches are calculated based on the IP addresses or cookies.

The attackers can stay under the radar level by giving up some greed, and increasing the number of publishers in the coalition. A group of attacking publishers, of size G, can make hard-to-detect coalitions by structuring their pages so that whenever a surfer visits any site, it automatically loads the sites of only a small number, g, of the publishers in the group, chosen randomly or in a round robin fashion. Assuming similar traffic volumes among sites, and that a site loaded inside a 0-size frame does not load another site, loading a limited number of sites per surfer's visit, rather than the sites of all the publishers, reduces the pairs' similarities from 1 to $\frac{g}{G-1}$, while keeping the gain factor at $g+1$. Hence, if the attacking group has to keep its inter-similarity at an undetectable level, s, the gain of each publisher in the group is $s(G-1)+1$. By forming big groups of attackers, attackers can suppress their inter-similarity, while still making gains

Increasing the size of the coalitions from pairs to large groups shifts our focus from searching for pairs of sites with similar traffic to searching for groups with similar traffic. The relationship of the sites can be modeled as an undirected graph, where nodes represent sites, and an edge between two sites represent a similarity of at least s. Hence, instead of just looking for edges, the objective changes to looking for densely connected subgraphs in a huge graph. This problem has been raised recently in the setting of search engines to model *Link spam* [8, 9, 10, 11], where the attackers link their sites in a way to mislead the connectivity-based PageRank ranking algorithm [27], and hence, increase the ranking of certain sites.

The algorithm presented in [9], which is based on the sketch-based *Shingling* algorithm [4], efficiently extracts dense subgraphs of sites to detect *Link spam*. The algorithm applies the *Shingling* algorithm to discover sites with common links, and produces, for each link, l_i, a set of sites, S_{l_i}, that share this link, and for each site, s_j, a set of links, L_{s_j}, that share this site. Then, it applies the *Shingling* algorithm recursively on the site-lists, $S_{l_i} \forall i$, to discover groups of links shared by groups of sites. Because of the sketching nature of the algorithm, each site-group might not necessarily contain all the sites that share this link-group. So, it merges those groups that share at least one set, L_{s_j}, for any j.

The real nightmare happens when the publishers attack more than one commissioner. For instance, if each publisher deals with a different commissioner, then each commissioner can only know the traffic of its publisher, and no commissioners can detect the similarity between the traffics of the attackers. The assumption that the attacking publishers belong to different commissioners made the attack in [1] very difficult to detect. In such cases, a third party should be hired that can see across the boundaries of the commissioners. In our solution [22] to the attack in [1], the ISPs played that role, since they see traffic that

```
Algorithm: Fast-Click-Daemon(Old-Imps, New-Imps, Imps-Buffer)
begin
  while True{
    //Moving Impressions from Imps-Buffer to New-Imps
    for each ID_I ∈ Imps-Buffer |(CurrentTime − timestamp_I) ≤ 1{
      for (Integer i = 1, i ≤ d, i++){
        Integer hash = f_i(e);
        New-Imps[hash]++;
      }// end for
    }// end for
    //Moving Impressions from New-Imps to Old-Imps
    for each ID_I ∈ Imps-Buffer |1 < (CurrentTime − timestamp_I) ≤ τ{
      for (Integer i = 1, i ≤ d, i++){
        Integer hash = f_i(e);
        New-Imps[hash]−−;
        Old-Imps[hash]++;
      }// end for
    }// end for
    //Deleting Impressions from Old-Imps
    for each ID_I ∈ Imps-Buffer |τ < (CurrentTime − timestamp_I) ≤ ϒ{
      for (Integer i = 1, i ≤ d, i++){
        Integer hash = f_i(e);
        Old-Imps[hash]−−;
      }// end for
    }// end for
  }// end while
end;
```

Figure 6: The *Fast-Click-Daemon* Algorithm.

```
Algorithm: Fast-Click-Detector(Old-Imps, New-Imps)
begin
  for each click, (timestamp_J, ID_J), in the stream{
    Boolean fast = False;
    Boolean slow = True;
    for (Integer i = 1, i ≤ d, i++){
      Integer hash = f_i(ID_J);
      if (New-Imps[hash] = 0){
        //Click with ID_J cannot be a fast
        fast = False;
      }
      if (Old-Imps[hash] = 0){
        //Click with ID_J is either slow or has no impression
        slow = True;
      }
    }// end for
    if (fast = True){output ID_J as fast;}
    if (slow = True){output ID_J as having no impression;}
  }// end for
end;
```

Figure 7: The *Fast-Click-Detector* Algorithm.

the commissioners cannot see. We anticipate the development of new "detective" entities that can see the traffic of several commissioners, and thus, be able to detect *coalition* attacks across several commissioners.

4.3 Detecting Non-Human Behavior

Another orthogonal way to detect publishers' attacks is to look for automatically generated traffic. This approach can be used against both *coalition*, and *non-coalition* fraud attacks. However, classifying traffic as being robotic or not is a difficult task.

One way of discovering robotic traffic is by checking for periodic impressions or clicks. Discovering periodicity in temporal data has been studied in several contexts. The work done by Ma *et al.* [18] is the most applicable in our case. The work in [18] checks for time-based periodic events with on-segments and off-segments in noisy temporal data. The on-segments are time segments that have the pattern, while the off-segments are periods that do not have the pattern occurring periodically. Since an attacking publisher can use a script that generates traffic with randomized periodicity [15], the devised algorithm should be able to accommodate high noise. Excessive experimentation is needed to find appropriate values for the parameters of the periodicity and the noise. Here, we propose another novel measure for detecting robotic clicks.

Detecting Fast Clicks

A reasonable way to detect automated clicks is to check the time difference between a click and its corresponding impression. Scripts normally simulate a click just after loading the page. The commissioner needs to employ an algorithm that matches every click with its impression based on the IDs of the cookie, the site, and the advertisement. We assume the duplicate impressions and clicks are already purged, and therefore, we will not face the problem of finding more than one impression matching one click. The algorithm should alert

if the time difference is less than τ time units, where τ is the predefined minimum allowed delay. In practice, the minimum allowed delay is around 5 seconds. Formally, given a click entry $(timestamp_J, ID_J)$, and a impressions stream on the form $(timestamp_I, ID_I)$, it is required to check if the last impression with the same ID of the click has a $timestamp_I$, such that $(timestamp_J - timestamp_I) < \tau$.

A preliminary approximate solution is devised here. The algorithm should keep a data structure, *Old-Imps*, which carries all the impressions' IDs (and not the timestamps) that have been observed at least τ units ago. To limit the amount of history kept by the commissioner, we assume that impressions older than a specific threshold, Υ, will be purged. Another data structure, *New-Imps*, will be kept that carries all the impressions' IDs (and not the timestamps) that have been observed in the last τ time units. The purpose of keeping *Old-Imps* is to make sure that every click has a preceding impression. The purpose of keeping *New-Imps* is to know which impressions are not old enough to be joined with clicks. In addition, we have to maintain a sequential buffer, *Imps-Buffer*, of the impressions received in the last $\Upsilon + 1$ time units.

To maintain the integrity of the data structures, every time unit, new impressions that have arrived in the last time unit are moved from *Imps-Buffer* to *New-Imps*; impressions that are becoming older than τ time units are deleted from *New-Imps*, and added to *Old-Imps*; and impressions that become older than Υ units are purged from *Old-Imps*. In order to know which impressions to move from *New-Imps* to *Old-Imps*, and which impressions to purge from *Old-Imps*, *Imps-Buffer* has to store the impressions sequentially.

When a click is received, it is only accounted for if its impression does not belong to *New-Imps* but exists in *Old-Imps*. That is, the impression with the same click identification (same cookie, site, and advertisement IDs) has been seen more than τ time units before the click.

The only remaining problem is devising a data structure that allows for constant time search, insertion, and deletion to cope with the stream processing constraints. The same data structure can be used for both *Old-Imps*, and *New-Imps*, since the same operations need to be supported by both data structures. We suggest using a vector of integers with d independent hash functions, like the one proposed for detecting u-duplicates in Section 4.1.2. Searching in this vector of integers entails hashing the click ID using the independent hash functions, and checking if all the corresponding integers are non-zero. Inserting an element entails hashing the impression ID using the independent hash functions, and incrementing all the corresponding integers. Deleting an element entails hashing the impression ID using the independent hash functions, and decrementing all the corresponding integers. An integer in *Old-Imps* will be 0 only if no impressions were inserted that hash to that integer, or all such impressions got deleted after becoming older than Υ units, and hence, their counters were decremented back to 0. A similar argument applies for *New-Imps*. The same hash functions can be used for both *Old-Imps* and *New-Imps* to reduce the hashing time. The daemon algorithm that runs in the background to maintain the data integrity is sketched in Figure 6.

The algorithm for detecting fast clicks is sketched in Figure 7. When observing a new click, its ID is hashed using the hash functions, and the click is only valid if at least one of the corresponding integers is 0 in *New-Imps*, and all the corresponding integers are non-zero in *Old-Imps*. The amortized time per impression handling is at most two searches, two insertions and two deletions, which is constant.

There are several approximations in the proposed solution. The errors in searching due to the hash collisions are the same as those analyzed in Section 4.1.2 for detecting u-duplicates, since a click is erroneously found to be fast if it is, by mistake, found to belong to *New-Imps*. There is another source of error caused by deleting impressions that are older than Υ from *Old-Imps*. Hence, Υ should be chosen wisely to balance the space usage, and the number of clicks lost due to their impressions becoming older than Υ units. A third source of error is that the impressions are inserted and deleted from the data structures only at the boundary of the time units. This can cause some errors. For instance, just after half a time unit of moving impressions older than τ units to *Old-Imps*, there will be some impressions' IDs in *New-Imps* that are $(\tau + \frac{1}{2})$ units old. The clicks of such impressions get discharged if they arrived before moving those impressions to *Old-Imps*. A straightforward solution, which increases the processing overhead, is to reduce the size of the time units whose boundaries trigger this impression-moving process.

5 The Proposed Architectural Model for Fraud Detection

Having discussed the classes of *non-coalition* and *coalition* fraud and detection techniques, we discuss how those techniques can be deployed in a real clustered Web server architecture. We start by describing a generic Web server architecture borrowed from [6], as well as discussing our assumptions in Section 5.1. Then, we show how the fraud detection techniques can be deployed on this generic architecture in Section 5.2.

5.1 The Generic Clustered Web Server Architectural Model

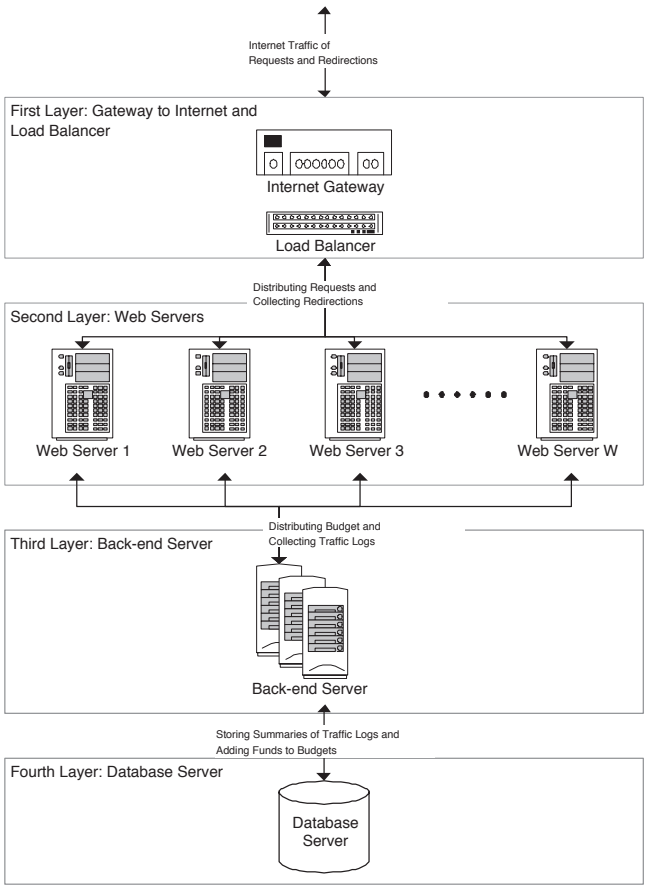


Figure 8: The Advertising Commissioners' Generic Web Server Architectural Model.

As illustrated in Figure 8, the architecture consists of four layers. As far as the fraud detection application is concerned, the first layer comprises the gateway to the Internet, through which the requests come from the Internet Browsers to the commissioner servers; as well as the load balancer, which assigns requests to different Web servers in the second layer of Web servers. We do not make any stickiness assumptions. That is, two consecutive requests from the same site, and same cookie ID can be routed to two different Web servers according to the load of the servers at the time the requests are made. As will be clear later, this assumption increases the portability of our model, though limits the optimizations possible.

The second layer comprises the Web servers. The primary functionality of the Web servers is serving the advertisements to the requests, redirecting clicks, logging traffic, and enforcing constraints like advertisers' geographical targeting, budgeting, etc. We assume the traffic logs collected by the Web servers are sent periodically to the third layer. The third layer is the Back-end that collects the data from the Web servers,

does some data manipulation, and re-adjusts advertisements' budgets that are being consumed by the Web servers. This layer is assumed to have several machines with a single file system, or is otherwise running on a single powerful machine with some redundancy for fault-tolerance. The fourth layer is the database layer, which is responsible for accounting, trend analysis, etc.

The architectural model discussed in this section is generic for advertising commissioners. Depending on the commissioner's business model, and scale, two or more of these layers can be merged together.

5.2 Deploying the Techniques on the Architecture

In this section, we discuss how to map the fraud detection techniques of Section 4 onto the architecture of Section 5.1.

5.2.1 Fraud Detection at the Second Layer

The Web servers do not have the complete picture of traffic across all sites or surfers. The fraud filters that can be used at this level are limited. In addition, it is not wise to use the CPU cycles of those machines for heavy fraud detection schemes, since they need to process requests in real-time. Fraud detection in this layer can filter out cookies that appear from a number of IP addresses greater than the commissioner threshold, p, in the last specified time period, t.

If stickiness is enforced, further optimizations are foreseeable. If the request is directed from the load balancer to the Web server according to a hash of the cookie ID, then all requests from the same cookie ID would end up on the same server. This is called *cookie-stickiness*. Hence, this layer could detect u-duplicates, *cookie-replay* attacks, and automated fast clicks. However, it is generally not recommended to balance the load among the Web servers according to a hash of the IP addresses, the advertisements, or the sites, since the data gets highly skewed when aggregated by these dimensions, which will lead to unbalanced loads.

5.2.2 Fraud Detection at the Third Layer

The Back-end server has a complete picture of the traffic. However, since it interacts with the real-time Web servers, it is not recommended to load this layer with time-consuming queries. Hence, only online stream processing that requires a single pass on the data should be deployed at this layer. Fraud detection filters that can be employed here are those that detect duplicates, *cookie-replay* attacks, and sites with high percentage of cookie-less traffic, as well as unbalanced activities of IP addresses and publishers (if a streaming solution is proposed in the future) and periodic traffic detection. This is also the right layer for an online algorithm that detects *coalition* attacks by finding high similarity between publishers' traffics.

5.2.3 Fraud Detection at the Fourth Layer

The database layer has a high level picture of the whole traffic. Commissioners take advantage of their knowledge of the dynamics of the entire system, while fraudulent publishers do not possess such knowledge. In addition, the support of a DBMS, and powerful machines are assumed. In this layer, highly analytical old-fashioned algorithms should be employed, whose results should compliment those of the streaming algorithms running on higher layers.

This layer is the best layer to deploy the defense mechanisms suggested in [15]. Those defense mechanisms include heavy SQL statements that would identify the sites with suspicious *Click Through Rates* (*CTR*)[6]. Some publishers try to generate clicks on their sites such that the *CTR* stays as a small percentage,

[6] *The Click Through Rate* of a publisher is the number of clicks it receives as a percentage of impressions.

say 1%, of the number of automated visits. In general, each advertisement have different *CTR*s depending on the quality of the advertisement, the advertised product, and most important, the relevance to the page. For instance, an advertisement about sports equipment is expected to have a higher ratio of clicks to impressions on a biking page than another advertisement about mortgages on the same page. Since the *CTR* of an advertisement is roughly consistent across sites of the same nature, it is informative to analyze the average *CTR* on each advertisement. Publishers's sites that deviate considerably from the average *CTR*, on other similar sites, of the same advertisements are suspicious.

The complementary observation is to run queries that detect sites that have the same *CTR* across different advertisements. Since the *CTR* differs from one advertisement to another, it is unlikely that different advertisements have the same *CTR* on the same site. Hence, such traffic is likely to be fraudulent.

The aforementioned query techniques running on the fourth (database) layer are usually effective for advertising networks because the commissioners use *dynamic placement*. Since an advertisement is displayed on several sites, and a site displays several advertisements at different times, the comparison of different advertisements on the same site or different sites on the same advertisement is revealing.

6 Case Study and Real Network Findings

In this section, we describe our experience with building a fraud detection system at Fastclick, Inc., a ValueClick company. First, we should point out that Fastclick does not contract with sites carrying pornographic materials[7]. Hence, the traffic of Fastclick is relatively clean. As a guide, we implemented some off-line mechanisms that measure the performance of publishers' sites. Since some of those mechanisms were expensive to calculate, we applied them for publishers with large traffics in order not to miss strong attacks, if any. The results suggested good parameter values for online algorithms.

6.1 Off-line Measures Set the Online Parameters

The most effective off-line metrics used was the *Click Through Rate* (*CTR*). As noted in Section 5.2.3, *CTR*s that deviate significantly from the rest of the network, on the same advertisements, are suspicious, since the *CTR* of an advertisement is roughly consistent across sites of the same nature. Similarly, publishers who have consistent *CTR*s across all advertisements are suspicious. Since it is difficult for publishers to know the *CTR*s of individual advertisements, this measure is effective against automatic traffic generation.

The second off-line metric was the *CPA-Score*. *Cost-Per-Action* (*CPA*) campaigns are those that pay the publishers according to the number of sales or actions made. Most of the *CPA* campaigns are *arbitrage campaigns*, where the publisher is paid according to the traffic volume, but the advertiser pays the commissioner according to the sales volume, as discussed in Section 2.2. The *CPA-Score* of a publisher is the profit margin of the commissioner on the traffic of this publisher. A *CPA-Score* of 0 means that the publisher costed the commissioner as much as it generated revenue. Since the commissioner has a good percentage *arbitrage campaigns*, and uses *dynamic advertisements placement*, it is fair to compare publishers using this measure.

The *Rich Media Click Rate* (*RMCR*) of a publisher is the number of its clicks on animated (rich-media) advertisements as a percentage of clicks on both animated and static (image) advertisements. There is a stable network-wide ratio between clicks on animated and clicks on image advertisements. Since it is technically more difficult to simulate clicks on animated advertisements, since the script has to parse the advertisement, this ratio gets violated if the publisher forges its clicks.

We used these off-line expensive measures to judge the performance of publishers, and hence were able to set the appropriate parameters for the online techniques discussed in Section 4.

[7]The reader is referred to http://fastclick.com/publishers/specifications.html.

It is crucial to note here that those measures cannot be used as a strong basis for contract termination. For instance, some non-professional sites have poor content, which affects the sales rate (*Conversion Rate*), or the *CTR*. Other sites, e.g. kids and games sites, have a very consistent high *CTR* across all advertisements. The reason is that a lot of the games' clicks, by mistake, go to the advertisements. Even more, some sites request only non-rich-media advertisements, since they are faster to load on the surfers' Browsers. This deviates the *RMCR* of such sites from the rest of the network.

6.2 Discovered Fraud Instances

After implementing, deploying, and tuning the defenses in Section 4, we were able to justify a lot of the poor measures of some publishers. We discovered four major kinds of outbreaks: automated repeated visits with the same identification, fast clicks, a cookie-replay attack, and the subtlest was a coalition attack.

Some fraudulent publishers employed simple scripts to generate automated visits to their sites. Some sites received more than 200 visits per hour from one pair of an IP address and a cookie ID. Other publishers had a high percentage of their clicks being generated within very few seconds of their corresponding impressions. The traffic that was generated using those simple attacks was less than 3% of the total traffic.

The cookie-replay attack was more interesting. Commissioners store the most recently displayed advertisements on the cookie to avoid showing repetitive advertisements and to guarantee good exposure for advertisers. Every time a request for an impression is received, the commissioner displays the most profitable advertisement that was not shown in the recent history, as described in Section 2.1. Thus, the picked advertisement depends on the recently shown advertisements, and the returns of all advertisements. Some publishers kept sending the same cookies repeatedly, which violated the *frequency-capping* rules, and was detected through the unchanged cookie counter. In the beginning, we thought these could be false positives of the Bloom Filters, but after we investigated the traffic database, it was clear that several publishers were receiving approximately 18% of their traffic without altering the cookies, even though most of those publishers have exemplary off-line ratings. The attack was not detected as a single IP address single cookie attack, since it was generated from various IP addresses, but with the same cookie IDs. This made us suspicious that the attacker wants the commissioner to display roughly the same advertisement every time. Hence, we became almost sure that the traffic is generated by competitive advertisers to deplete the budgets of their competitors. Although this traffic was directed to a few advertisers, it summed up to 1% of the traffic, and hence depleted the advertisers budget rapidly.

The discovered coalition attack was the subtlest. We analyzed a data sample of 54,045,873 traffic entries for site pairs of similar traffic using the *Shingling* algorithm [4]. To reduce the noise, we excluded all the IP addresses visiting a large number of sites, because such IP addresses are probably coming from NAT boxes. We repeated the experiment and progressively increased the shared number of sites, beyond which the IP addresses are disregarded, from 10 to 1,000. The results are plotted in Figure 9, with a logarithmic scale on the vertical axis[8]. When the number of sites, beyond which the IP addresses are disregarded, was 1,000, 98.94% of the pairs had less than 1% of similarity. When it was 10, 99.98% of the pairs had less than 1% of similarity. For each run, the algorithm output all the pairs with similarity more than 10%, and we fed the output to the clustering algorithm [9] to discover groups of sites whose IP addresses overlap largely.

When the number of sites, beyond which the IP addresses are disregarded, was 10, the clustering algorithm output several groups, each of size 3 to 5 sites. As we increased the number of sites, beyond which the IP addresses are disregarded, the clustering algorithm was able to connect most of these components into bigger groups. The reason is that the algorithm considered IP addresses that are shared by these disjoint components. When this number reached 30, there were three groups with high inter-similarity of sizes 3, 3, and 29. As the number grew, the clustering algorithm did not connect those components, but

[8] 1 was added to each bar in the graph to have a logarithmic scale on the vertical axis.

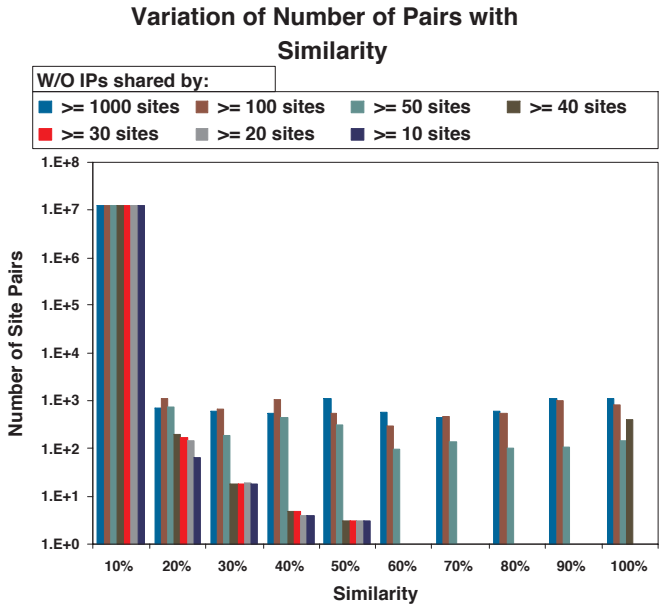


Figure 9: The Variation of the Number of Pairs (Logarithmic Scale) as the Similarity Changes.

rather started to output groups that share extremely popular IP addresses. We checked those IP addresses on www.arin.net/whois, and it was clear that they are ISP-owned IP addresses.

The sites looked fine, except that their traffics were of moderate size, yet coming from IP addresses all over the world. The strange thing was 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[9], the publishers, somehow, acquired the attached activation secrets, and activated the accounts. Since no employees know the activation secrets, the attackers must have compromised some machines on those domains, and hence, were able to know the attached secrets. Since we only checked sites with high traffic in our off-line metrics, those sites were not detected by any off-line mechanism.

7 Conclusion

In this paper, we have described an environment, the advertising networks, which is rich with research problems. We provided a classification of the publishers' *non-coalition* and *coalition* fraud techniques. In addition, we proposed several techniques for detecting automated traffic. We developed stream-based solutions for most of the problems, and abstracted some theoretical stream analysis problems for the research community. In addition, we discussed how fraud detection algorithms can be deployed on a generic architecture. Finally, we communicated our experience in building a fraud detection mechanism on a real network. A highly challenging open problem is finding active pairs of publishers and IP addresses that are considered anomalous. There are several ways to specify anomalous pairs in a two-dimensional stream, where the two dimensions are the IP address and the publisher. The scale of the problem is huge since both dimensions

[9]When a publisher signs up with a commissioner, the commissioner 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.

are vast. Hence, the space required for solving the problem has to scale for the current size. In addition, the streaming nature of the problem imposes tight constrains on the processing time per element.

The more intriguing problem is detecting attacks with several cookies and several IP addresses. Formalizing the problem in a tractable way is a challenge, and solving the problem is expectedly more difficult. The area is still open for other classifications of *hit inflation* attacks, new techniques to hunt for automated behavior, and more effective solutions for the problems proposed here.

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 International Conference on World Wide Web*, pages 1091–1100, 1999.

[2] Burton H. 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, S. Glassman, and M. Manasse. Syntactic Clustering of the Web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166, 1997.

[5] E. Burns. Online Seizes More of the Advertising Mix. *ClickZ News*, February 13 2006.

[6] V. Cardellini, E. Casalicchio, M. Colajanni, and P. Yu. The State of the Art in Locally Distributed Web-Server Systems. *ACM Computing Surveys*, 34(2):263–311, 2002.

[7] 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.

[8] D. Fetterly, M. Manasse, and M. Najork. Spam, Damn Spam, and Statistics: Using Statistical Analysis to Locate Spam Web Pages. In *Proceedings of the 7th WebDB International Workshop on the Web and Databases*, pages 1–6, 2004.

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

[10] Z. Gyöngyi and H. Garcia-Molina. Link Spam Alliances. In *Proceedings of the 31st VLDB International Conference on Very Large Data Bases*, pages 517–528, 2005.

[11] M. Henzinger, R. Motwani, and C. Silverstein. Challenges in Web Search Engines. *ACM SIGIR Forum*, 36(2):11–22, 2002.

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

[13] R. Khare and A. Rifkin. Trust management on the World Wide Web. In *Proceedings of the 7th WWW international conference on World Wide Web*, pages 651–653, 1998.

[14] V. Khu-smith and C. Mitchell. Enhancing the Security of Cookies. In *Proceedings of the 4th ICISC International Conference on Information Security and Cryptology*, pages 132–145, 2001.

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

[16] P. Lerma. When Should Online Come Before TV? *ClickZ News*, September 20 2005.

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

[18] S. Ma and J. Hellerstein. Mining Partially Periodic Event Patterns with Unknown Periods. In *Proceedings of the 17th IEEE ICDE International Conference on Data Engineering*, pages 205–214, 2001.

[19] A. Marcus. Branding 101. *interactions*, 11(5):14–21, 2004.

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

[21] 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. An extended version appears as a University of California, Santa Barbara, Department of Computer Science, Technical Report 2004-23.

[22] 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. An extended version appeared as a University of California, Santa Barbara, Department of Computer Science, technical report 2005-13.

[23] R. Miller and K. Bharat. SPHINX: A Framework for Creating Personal, Site-Specific Web Crawlers. In *Proceedings of the 7th International Conference on World Wide Web*, pages 119–130, 1998.

[24] D. Morgan. Making the Cookie Case to Consumers. *ClickZ News*, March 24 2005.

[25] 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.

[26] T. Novak and D. Hoffman. New Metrics for New Media: Toward the Development of Web Measurement Standards. *World Wide Web Journal*, 2(1):213–246, 1997.

[27] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford University, 1998.

[28] 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.

[29] Z. Rodgers. Revived Spyware Bill Could Crunch Cookies. *ClickZ News*, January 6 2005.

[30] S. Sariou, S. Gribble, and H. Levy. Measurement and Analysis of Spyware in a University Environment. In *Proceedings of the 1st ACM/USENIX Symposium on Networked Systems Design and Implementation*, pages 141–153, 2004.

[31] C. Schroeder. Bottom-Feeders or an Advertising Revolution? *Media Post Publications, MediaDailyNews*, November 23 2004.

[32] G. Shaw. Spyware & Adware: the Risks Facing Businesses. *Network Security*, 2003(9):12–14, 2003.

[33] J. Thaw. Online Ad Growth Accelerates, Outpacing Newspaper, TV Spending. *Bloomberg News*, December 28 2005.

[34] H. Thomases. The Rise of CPA Ad Networks. *ClickZ News*, September 13 2005.
[35] D. Vise. Clicking To Steal. *Washington Post Magazine*, page F01, April 17 2005.