Detecting Click Fraud in Pay-Per-Click Streams of Online Advertising Networks

Linfeng Zhang and Yong Guan
Department of Electrical and Computer Engineering
Iowa State University, Ames, Iowa 50011
{zhanglf, guan}@iastate.edu

Abstract

With the rapid growth of the Internet, online advertisement plays a more and more important role in the advertising market. One of the current and widely used revenue models for online advertising involves charging for each click based on the popularity of keywords and the number of competing advertisers. This pay-per-click model leaves room for individuals or rival companies to generate false clicks (i.e., click fraud), which pose serious problems to the development of healthy online advertising market. To detect click fraud, an important issue is to detect duplicate clicks over decaying window models, such as jumping windows and sliding windows. Decaying window models can be very helpful in defining and determining click fraud. However, although there are available algorithms to detect duplicates, there is still a lack of practical and effective solutions to detect click fraud in pay-per-click streams over decaying window models.

In this paper, we address the problem of detecting duplicate clicks in pay-per-click streams over jumping windows and sliding windows, and are the first that propose two innovative algorithms that make only one pass over click streams and require significantly less *memory space and operations. GBF algorithm is built on* group Bloom filters *which can process click streams over jumping windows with small number of sub-windows, while TBF algorithm is based on a new data structure called* timing Bloom filter *that detects click fraud over sliding windows and jumping windows with large number of sub-windows. Both GBF algorithm and TBF algorithm have zero false negative. Furthermore, both theoretical analysis and experimental results show that our algorithms can achieve low false positive rate when detecting duplicate clicks in pay-per-click streams over jumping windows and sliding windows.*

1 Introduction

With the rapid growth of the Internet, online advertisement plays a more and more important role in the advertising market. Among several online advertising models, pay-per-click model is the most popular one. However, pay-per-click model is suffering serious fraud problems: attackers earn extra incomes or deplete competitors' advertising budget by simply clicking (seldom by hands, often by automated scripts or bots) the pay-per-click advertisements without actual interest in the content of the ad's link. Such fraudulent clicks not only exhaust online advertisers' money, but also destroy the trust between online advertisers and advertising publishers, and hence damage the healthiness of online advertising market. Recently, there are several class action lawsuits against large online advertising publishers. Therefore, the development of feasible and effective solutions to click fraud problems may benefit both the advertisers and the publishers.

An important issue in defending click fraud is how to deal with duplicate clicks. If we simply regard all identical clicks as fraudulent clicks, it is unfair to advertisers in some scenarios such as that an interested client visits the same ad link several times in a week. On the other hand, if the advertisers are charged for any identical clicks, then it is very easy for an attacker to make money by continuously clicking the same ad link. A reasonable tradeoff is to define a timing threshold and only count identical clicks once within the timing window. Decaying window models, such as landmark, jumping and sliding window models, are feasible to solve this problem. However, most traditional duplicate detecting algorithms may not be directly deployed to address this problem over decaying window models.

One possible solution is to use data streaming algorithms, which have received considerable attention recently [5, 24]. Many characteristics of large data streams, such as sum, mean, variance, frequency, quantile, top-k list (hot list), distribution, etc, have been widely studied. However, the problem of duplicate detection over different decaying window models still lacks efficient and effective solutions.

1063-6927/08 $25.00 © 2008 IEEE
DOI 10.1109/ICDCS.2008.98

In this paper, we will describe two effective and efficient algorithms to detect click fraud in pay-per-click streams over different kinds of decaying windows, while using as little space and operation as possible and making only one pass over the click streams.

1.1 Motivation

Although online advertising is an infant comparing with traditional advertising media, it grows very quickly and plays a more and more important role in the advertising market. Several studies show that more than 10 billion dollars are spent in online advertising market annually [11, 17]. Among several online advertising models, such as pay-per-action, pay-per-call, pay-per-click, etc, pay-per-click model is the most popular one. Online advertisers bid on keywords of search engines or ad links of online publishers such that their target links can have more chance to be visited by end users. However, click fraud problem heavily challenges the pay-per-click advertising model: the ad link is clicked without actual advertising impression. A survey indicates that Internet advertisers paid \$0.8 billion for click fraud in 2005 and \$1.3 billion estimated in 2006, and about 14.6% clicks are fraudulent [25].

The possible source of click fraud may be: 1. Search engines or online ad publishers themselves; 2. Ad subdistributors; 3. Competitors; 4. Web page crawlers. Fraudulent clicks can be produced by hands, by automated scripts, or by botnets, etc. Several types of click fraud, such as hit shaving problem [26] and hit inflation attacks [3], etc, have been studied, and several algorithms have been proposed to prevent such types of click fraud. However, many of them can only be deployed by the advertising publishers. Not all advertising publishers have enough motivations to deploy these algorithms, since they receive money for each click, even if it is fraudulent. Therefore, there is a conflict in detecting click fraud: Online advertisers have enough motivations to prevent click fraud but little abilities; Online advertising publishers have more power to play a decent role in defending click fraud but without enough incentives. Such a conflict may lead to distrust between online advertisers and publishers. Recently, several class action lawsuits against large online advertising publishers appear. Google paid 90 million dollars to settle a class action lawsuit about click fraud in March 2006 [1]. In June 2006, Yahoo settled a similar lawsuit by paying 4.95 million dollars to plaintiffs' counsel, and allowing credit refund to advertisers who claim click fraud back through January 2004 [2].

A possible solution is that both the online advertisers and publishers keep on auditing the click stream and reach an agreement on the determination of valid clicks. When determining which are valid clicks in the click streams, an important issue is how to define *duplicate clicks*. Should an advertiser be charged once or twice when there are two identical clicks? Let us consider the following two scenarios.

Scenario 1: A normal client visited an advertiser's web site by clicking the ad link of a publisher. One week later, the client visited the same web site again by clicking the same ad link.

Scenario 2: The competitors or even the publishers control a botnet with thousands of computers, each of which initiate many clicks to the ad links everyday.

Obviously, the clicks in Scenario 1 should not be considered as click fraud, while those in Scenario 2 should be determined as click fraud. However, it is very difficult to identify which scenario the identical clicks belong to. A reasonable countermeasure is to prescribe that identical clicks will not count if they are within short time interval, and will count if they happen sparsely. Therefore, a feasible duplicate detecting algorithm should have a mechanism that is able to eliminate expired data and only consider fresh information. Decaying window models are feasible to be utilized to eliminate expired information. Although recently many algorithms are proposed for capturing different kinds of characteristics of large data streams over decaying window models [4, 6, 9, 15, 16, 18, 19, 27, 29, 28, 31], the problem of duplicate detection in data streams over decaying window models still lacks efficient and effective solutions. In the following, we will discuss a number of useful decaying window models.

1.2 Decaying Window Models

Decaying window models can be utilized to eliminate expired information. There are two common types of decaying windows: *count-based* windows which maintain the last (most recent) N items in the data stream, and *time-based* windows which maintain all items that arrived in the last T time units. Therefore, the time span of a count-based window may vary, while the number of items in a time-based window may change from time to time. In this paper, we mainly consider count-based windows, and our algorithms can be easily extended to time-based windows. How to extend to time-based windows can be found in Section 3.1 and 4.1.

Landmark window: Landmark windows start and end once N elements arrive. When processing data streams over landmark windows, a sketch can be maintained using less memory instead of keeping all elements in current window, since all elements will expire at the same time and we can delete the expired sketch and begin a new one.

Sliding window: A sliding window, first introduced by Datar et al. [9], only contains the last N items, which is updated once a new element comes and an old element expires. Since the elements in a sliding window expire one

by one, usually some timing information is maintained to update the interested statistics when the window slides.

Jumping window: The jumping window model was first proposed by Zhu and Shasha [31]. A jumping window is a compromise between the landmark window and the sliding window. The baseline idea is to divide the entire window equally into several sub-windows, and the statistics over the entire jumping window is based on the combination of the information from the smaller sub-windows.

1.3 Problem Statement

We first give the definition of a duplicate click in pay-per-click streams over decaying window models.

Definition 1 *A click is classified as a* duplicate click *if in the current decaying window an identical click has been determined as a valid click.*

In this paper, we consider the problem stated as follows:

Given limited memory and an arbitrary window size N (N elements or N time unites), how to effectively and efficiently detect duplicates in a click stream over jumping windows or sliding windows in one pass?

1.4 Our Contributions

In this paper, we are the first that address the problem of detecting duplicate clicks in pay-per-click streams over jumping windows and sliding windows using *significantly less* memory space and operations. Since a naive deployment of classical Bloom filters to jumping windows requires many memory operations, we propose an innovative GBF algorithm using group Bloom filters which significantly reduces the memory operations when processing click streams. Our GBF algorithm is effective and efficient over jumping windows with small number of sub-windows.

However, in a jumping window when there are too many sub-windows, GBF algorithm still requires many memory operations. To solve this problem, we propose a new data structure called timing Bloom filter, which records inserted elements' timing information. We design a TBF algorithm based on timing Bloom filter which can process click streams over sliding windows and jumping windows using less memory space and processing time.

One advantage of our GBF algorithm and TBF algorithm is that both of them have no false negative. Furthermore, both theoretical analysis and experimental results show that our algorithms are effective and efficient which can achieve low or bounded false positive rate when detecting duplicate clicks in pay-per-click streams over jumping windows and sliding windows.

Paper Organization

Section 2 discusses the related work. We first consider the problem of detecting duplicate clicks over jumping windows, and introduce group Bloom filters and GBF algorithm in Section 3. To detect duplicate clicks over sliding windows, timing Bloom filters and TBF algorithm are introduced in Section 4. We evaluate our algorithms with synthetic experiments in Section 5. We finally conclude and discuss our future work in Section 6.

2 Related Work

2.1 Classical Bloom Filters

Bloom filters were presented by Burton H. Bloom [7] in 1970, and have been widely utilized in many areas such as networking and database [8]. A Bloom filter is a space-efficient data structure for representing a set of n elements to respond membership queries. It is a vector of m bits which are all initialized to value 0. Suppose that we have a dataset of n elements. Then each element is inserted into the Bloom filter by hashing it using k independent uniform hash functions with range $\{1, 2, ..., m\}$ and setting the corresponding k bits (some bits may be overlapped) in the vector to value 1. Given a query whether an element is present in the Bloom filter, this element is hashed using the same k hash functions and the Bloom filter is checked whether all the corresponding bits are set to 1. If any one of them is 0, then undoubtedly this element is not in the dataset; otherwise, we would say that it is present in the dataset, although there is a certain probability that the element is falsely determined to be in the dataset while it is actually not. Such false cases are called *false positives*. The space-efficiency of Bloom filters is achieved at the cost of small acceptable false positive rate f. Suppose that n distinct elements are inserted into the Bloom filter, from [12], the false positive rate is $f = (1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-\frac{kn}{m}})^k$. When m and n are given, f is minimized when $k = \ln 2 \times \frac{m}{n}$. Thus, we have $f \approx 2^{-k} \approx 0.6185^{\frac{m}{n}}$. Bloom filters have been utilized to detect duplicates in databases. Furthermore, Bloom filters have been widely applied and modified to many other network and database problems. A detailed survey of network applications of Bloom filters was presented in [8].

2.2 Duplicate Detection

Besides the Bloom filter-based duplicate detecting algorithms, many other algorithms have been proposed in different research areas such as database systems, operating systems, computer architecture and network security, etc. The problem of exact duplicate detection is well studied, and many algorithms have been proposed. Please refer to [14] for more relevant references.

2.3 Data Stream Techniques

The problem of capturing characteristics of large data streams has received considerable attention recently [5, 24]. If there is sufficient large space and no time constraints exist, we can obtain any stream characteristics that we want precisely. However, the issue in processing large data streams is that in many cases we probably do not have enough space to keep each element in the data streams. An even greater challenge is to process data streams over decaying window models. Datar et al. [9] proposed an algorithm to solve the *Bit-Counting* problem over sliding windows using *Exponential Histograms* (EH). Besides the bit-counting problem [9, 15], several other problems of capturing characteristics of data streams over decaying window models are studied [4, 6, 16, 18, 19, 27, 29, 28]. Zhu and Shasha [31] introduced *basic windows* (i.e., jumping windows) to compute the statistics.

2.4 Online Advertising Fraud Detection

Reiter et al. studied the *hit shaving* problem [26]. They considered the referral revenue model that the referrers get payment for every click-through to target sites. A hit shaving is a practice that the target site undetectably omits to pay a referrer for some number of clicks.

Anupam et al. proposed a *hit inflation* attack on pay-per-click online advertising schemes [3]. Hit inflation is a kind of click fraud which is difficult to detect, where a referrer artificially inflate the customers' clicks to make illegal profit. Metwally et al. proposed an algorithm called *Streaming-Rules* to detect hit inflations in data streams [22]. They also built an algorithm called *Similarity-Seeker* to discover coalitions among advertising publishers [20], and a framework is outlined on a generic architecture [23].

Metwally et al. considered the problem of detecting duplicates in click streams [21]. They proposed a scheme over landmark windows which is a direct deployment of Bloom filters [7]. They also discussed how to detect duplicates over jump windows and sliding windows. To run over sliding windows, they use a modification of Bloom filters named *Counting Bloom Filter*, which is similar to [12]. However, their solution must keep all active click identifications in memory to slide them out later after they expire.

Deng and Rafiei considered the problem of approximately eliminating duplicates in streams with a limited space [10]. They proposed a data structure named *Stable Bloom Filter*, which randomly evicts the stale information to release room for more recent elements. However, their randomly evicting mechanism introduces false negatives besides the inherent false positives of Bloom filters.

Gandhi et al. described a type of click fraud threat to Internet advertising called *badvertisement* [13]. This attack utilizes malicious JavaScript to publish sponsored advertisements on clients' web browsers invisibly. The authors proposed active and passive approaches to detect and prevent such kind of click fraud.

3 Detecting Duplicates over Jumping Windows Using Group Bloom Filters

3.1 GBF Algorithm Description

To detect duplicates in click streams over a landmark window, Bloom filters can be directly deployed [21]. Each click has an predefined identifier, such as the source IP address, or the cookie, etc. Then each click's identifier is hashed into the Bloom filter. If a click's identifier is present in the Bloom filter before insertion (i.e., all corresponding bits are 1s), then it is reported as a duplicate.

To detect duplicates in pay-per-click streams over jumping windows, a naive solution is to evenly divide the entire jumping window into a number of sub-windows and maintain a separate Bloom filter for each sub-window. To save operation time, all Bloom filters should use the same set of hash functions. Suppose N is the size of the jumping window which is divided into Q sub-windows. After the jumping window is full, there will be an expired Bloom filter after each $\frac{N}{Q}$ elements. Therefore, if we want to use the memory space of the expired Bloom filter for the upcoming sub-window, we must clean the entire expired Bloom filter before the first element of the upcoming sub-window can be inserted. However, considering that cleaning an expired Bloom filter need $O(m)$ operations, where m is the size of the Bloom filters, we must keep the newly arrived elements in an extra queue before we finish the clean operations. To solve this problem, we can divide the total available memory space into $Q + 1$ pieces. Q pieces are for the Bloom filters of Q active sub-windows, and the additional piece is used to maintain a Bloom filter for the elements in the upcoming sub-window while we clean the expired Bloom filter. Then we have more time to clean the expired Bloom filter, since we only need to clean the expired memory before the next Bloom filter expires. Let M denote the total number of memory bits, then each Bloom filter has size $m = \frac{M}{Q+1}$, and we only need to clean $\frac{M/(Q+1)}{N/Q} = \frac{QM}{(Q+1)N}$ bits when processing each newly arrived element.

When a new element comes, it is inserted into the Bloom filter of the corresponding sub-window if and only if it is not a duplicate in the current jumping window (including all active sub-windows). To check whether it is a duplicate, we first calculate k hash values using the element's identifier. We then have to check each of the Q active Bloom filters (suppose the jumping window is full) by reading the corresponding k bits. If the set of k bits in any Bloom filter is

all 1s, then this element is reported as a duplicate click; otherwise, the corresponding k bits in the Bloom filter of the current sub-window are set to 1.

Obviously, such a duplicate-checking procedure may cost about $(Q \times k)$ memory operations, which is very time consuming if Q is large. To solve this problem, we introduce a data structure called *Group Bloom Filters* (GBF) which can significantly reduce required memory operations. The baseline idea is that instead of dividing the entire memory into separate pieces for separate Bloom filters, the bits with the same index in each Bloom filter are grouped together in GBF. Then using this data structure, the CPU can visit the required bits in a bunch. For instance, suppose that $Q = 31$ and the size of a word in the memory is 32 bits. Then the same bits of the total 32 Bloom filters will be in the same word in the memory. Suppose that CPU can read/write one 32-bit word each time, then we can fetch all bits we need using k memory reads. After we get the k 32-bit words, we AND them into a single 32-bit word. We then mask the bit which represents the expired Bloom filter in the word by setting the corresponding bit to 0. If the value of this word is none-zero, then the new element is a duplicate; otherwise, we set each corresponding bit for current sub-window in the k 32-bit words to 1 and write them back to the memory.

Extension to Handle Time-Based Windows

GBF algorithm can be easily extended to handle time-based jumping windows. Instead of dividing entire jumping window equally by counting elements, the time-based jumping window is divided into Q sub-windows with same time expansion. Then each sub-window is equally divided into R time units. In Step 1, the cleaning procedure executes once in each time unit, and scans $\frac{M}{(Q+1)R}$ entries. Step 2 stills executes once a new element comes.

3.2 Theoretical Analysis

When designing algorithms to detect duplicate clicks in pay-per-click streams over jumping windows, we must consider the false negative rate and false positive rate, the memory requirement, and the processing time. The following theorem provides the properties of our GBF algorithm, and its proof can be found in [30].

Theorem 1 *Let N denote the size of the jumping window, which is divided into Q sub-windows. Given M-bit memory, and assuming that the CPU can read/write a D-bit word in each cycle, group Bloom filters can detect duplicates over jumping windows with the following properties:*

1. *The false negative rate is 0.*

2. *The false positive rate is $O(Q \cdot 0.6185^{\frac{M}{N}})$.*

3. *The running time to process each element is $O(\frac{Q}{D} \cdot \frac{M}{N})$ in worst case.*

3.3 Comparison with Previous Work

In [21], the authors proposed to maintain a counting Bloom filter for each sub-window, and a main Bloom filter which is a combination of all counting Bloom filters and represents the entire jumping window. When a new sub-window is generated, the eldest window is expired and subtracted from the main Bloom filter. Combining two counting Bloom filters is performed by adding the corresponding counters; deleting an old counting Bloom filter is performed by subtracting its counters from the main Bloom filter.

However, this scheme has two potential drawbacks. One is that subtracting an expired Bloom filter from the main Bloom filter needs $O(m)$ operations, and false positives increase if new elements are inserted into the main Bloom filter before subtracting operation completes. The other drawback is that this scheme may have high false positive rate, especially when the number of sub-windows is large. There are two reasons for this drawback. First, with the same limited available memory space, expanding bits in Bloom filters to counters make the size of Bloom filters smaller. In worst case, the maximum value in the counters of counting Bloom filters is $\frac{N}{Q}$, and the maximum value in the counters of the main Bloom filter is N. Therefore, each counter must have enough bits to avoid saturation, which will generate both false negatives and false positives. Consequently, the size of the Bloom filters in their algorithm is much smaller than the size of Bloom filters in our GBF algorithm, and the false positive rate will be much higher than that of GBF algorithm. Second, checking the presence of an element in the main Bloom filter which is the result of combination of all counting Bloom filters will generate very high false positive rate, since it is as if all N elements are inserted into the single main Bloom filter (any entry with a non-zero value in any counting Bloom filter will set the corresponding entry in the main Bloom filter to non-zero). On the contrary, each Bloom filter in GBF algorithm is only inserted at most $\frac{N}{Q}$ elements.

Figure 1 shows the comparison between the algorithm in [21] and our GBF algorithm. We draw the false positive rate when $Q = 31$, $m = 2^{20}$, and N increases from 2^{15} to 2^{21}. We make the observation that with the increasing window size, the false positive rate of the algorithm in [21] increases more quickly compared with GBF algorithm when both algorithms maintain Bloom filters with the same size. For instance, when $N = 2^{20}$, the false positive rate of the algorithm in [21] is about 0.62, while the false positive rate of GBF algorithm is only about 0.000011. Notice that when both algorithms maintain Bloom filters with the same size, the algorithm in [21] requires more memory space since its Bloom filters are counting Bloom filters which contain more bits in each entry.

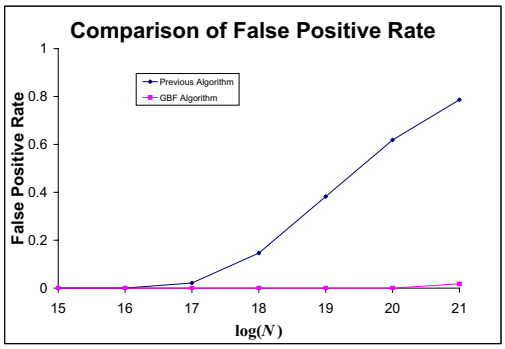


Figure 1. Comparison between Previous Algorithm and GBF Algorithm

4 Detecting Duplicates over Sliding Windows Using Timing Bloom Filters

GBF algorithm works well over jumping windows with small number of sub-windows. However, there is a limitation of GBF algorithm that it is not feasible in the sliding window model, or when the number of sub-windows Q is very large in a jumping window. In sliding windows, although we can keep N Bloom filters, each of which only hold one element, maintaining N Bloom filters make the running time unacceptable. For instance, suppose the window size $N = 2^{20}$, and the CPU can read/write 64 bits per cycle. Before an element is inserted into the Bloom filter for that sub-window, $16384 \cdot k$ reads must be executed, where k denotes the number of hash functions. Therefore, the GBF algorithm cannot process high-speed click streams over sliding windows or jumping windows with large number of sub-windows. We hence devise an innovative TBF algorithm based on a new data structure called *Timing Bloom Filters* (TBF) to detect duplicate clicks over sliding windows and jumping windows with large number of sub-windows.

4.1 TBF Algorithm Description

We propose a new data structure called timing Bloom filters which contain timing information derived from classical Bloom filters. The timing information contained in TBF can be utilized to evict stale data out of our data structure, and make TBF applicable to process data streams over sliding windows. Let N denote the sliding window size. An existing element is called *active* if it is one of the most recent N elements within the current window, or *expired* if it left the current window. For each element x_i, an index pos_i is used to record its timestamp (i.e., position) in the data stream, which is an indicator of "active" or "expired" by comparing with pos – the position index of the most recent element.

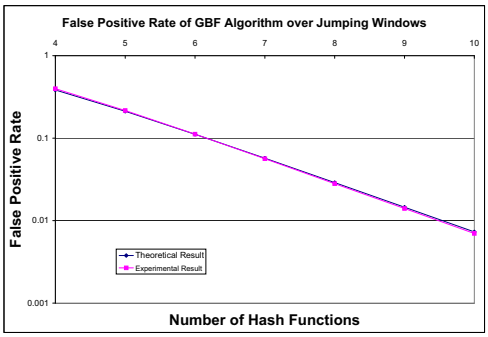
Our new data structure TBF is based on Bloom filters. To insert timing information into TBF, each bit in the classical Bloom filter is replaced by an entry with $O(\log N)$ bits. At the beginning of the click stream, all bits in all entries of the TBF are initialized to bit 1. When a new element arrives, we first calculate the k hash functions and get (at most) k indices. We then check the corresponding k entries to judge if this element is both present[1] in the TBF and active[2] in the current sliding window. If it is present and active, then we just ignore it and report it as a duplicate click; otherwise, we set the corresponding k entries using this element's timestamp. The timestamps are represented by wraparound counters, and the number of bits in each entry of TBF is set to be large enough such that no timestamp is represented by all 1s.

Our TBF algorithm has two steps when a new element arrives. Step 1 deletes expired information in TBF; Step 2 processes the new element. When a new element x_t comes, the current timestamp pos is updated. Usually there is an old element expired (except at the beginning of the click stream when the sliding window is not full). Therefore, besides processing the new arriving element, the expired timestamps in the TBF must be removed, which means TBF algorithm only maintains the timestamps of active elements in the current window. To bound the bits to represent the timestamp in the continuous click stream, we have to use a wraparound counter. We first consider the scenario that we use a wraparound counter with maximum $N-1$ to represent the timestamps, that is, the N-th element's timestamp is 0 instead of N. Suppose the newly arrived element's timestamp is P, then in this scenario the expired element also has timestamp P. Therefore, before inserting the new element into the TBF, we must first remove these expired timestamps if they have been inserted before. However, since the expired element is not maintained in memory, we have no knowledge about where the potential k expired timestamps are in the TBF. Consequently, we must scan the entire TBF with m entries to find these expired timestamps and replace them by all 1s. Since this is time consuming which needs $O(m)$ operations, such an algorithm cannot process high-speed click streams. We therefore propose an advanced update mechanism which only uses $O(\frac{m}{N})$ operations and only consumes very small additional space.

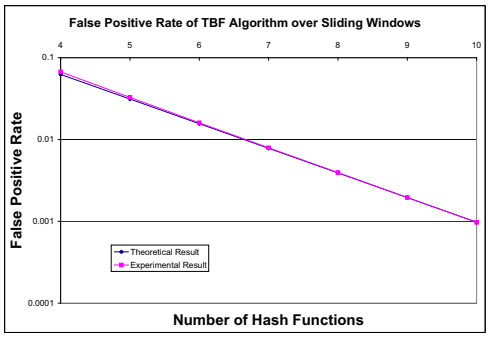
In our update mechanism, instead of setting $N - 1$ as the maximum value in the wraparound counter, we set $N + C - 1$ as the maximum timestamp, where C is an integer. Since now we expand the range of timestamp representation, we get extra time to remove stale timestamps and thus less entries in TBF need to be scanned each time. As discussed above, if $C = 0$, we have to scan entire m

[1] "Present" means no entry in the k corresponding entries is all 1s.

[2] "Active" means all timestamps in these k corresponding entries are within the current sliding window.



(a) False Positive Rate of GBF



(b) False Positive Rate of TBF

Figure 2. False Positive Rate of GBF and TBF Algorithm over Sliding Windows

entries when processing each new element. If $C = 1$, we only need to scan half of the m entries. Therefore, when a new element arrives, we only need to check $\frac{m}{C+1}$ entries in TBF. The entire TBF will be scanned thoroughly after $C+1$ elements arrive.

The choice of value of C is flexible. Since $\lceil \log(N + C + 1) \rceil$ is the number of bits in an entry to represent timestamps[3], and $\frac{m}{C}$ is the number of entries that need to be scanned per element to update the TBF, then a smaller C means less space requirement and larger operation time, and a larger C means larger space requirement and less operation time. In the following analysis and experiments, we typically choose C equal to $N - 1$.

Extension to Handle Time-Based Windows

TBF algorithm can be easily extended to handle time-based sliding windows. Suppose the entire sliding window is equally divided into R time units. In Step 1, the cleaning procedure executes once in each time unit, and scans $\frac{m}{R}$ entries. Step 2 stills executes once a newly arrived element comes. However, instead of inserting the counting-based position, the time unit information is inserted into the entries of TBF.

Furthermore, TBF can also be easily extended to handle jumping windows. If TBF is utilized over a jumping window which is evenly divided into Q sub-windows, then all elements in the same sub-window will have the same timestamp, and they will be eliminated from TBF simultaneously. When Q is large, GBF cannot process the click stream efficiently, and TBF is a better choice.

4.2 Theoretical Analysis

The following theorem provides the properties of our TBF algorithm when detecting duplicate clicks in pay-per-click streams over sliding windows. Its proof is also provided in [30].

[3]The additional one representation is for all 1s.

Theorem 2 *Let N denote the size of the sliding window. Given M-bit memory, timing Bloom filters can detect duplicates over sliding windows with the following properties:*

1. *The false negative rate is 0.*

2. *The false positive rate is $O(0.6185^{\frac{M}{N \log N}})$.*

3. *The running time to process each element is $O(\frac{M}{N \log N})$ in worst case.*

5 Experimental Evaluation

In this section, we evaluate GBF algorithm and TBF algorithm for duplicate click detection over jumping windows and sliding windows. Since our algorithms have no false negative, we only ran experiments to evaluate the false positive rate of our algorithms. Hence, we simulate our algorithms by processing synthetic click streams which have no duplicate click.

In the experiments of evaluating our GBF algorithm over jumping windows, we set the jumping window size to $N = 2^{20}$, and the number of sub-windows to $Q = 8$. We generated $20 \cdot N$ distinct click identifiers. We counted the false positives within the last $10 \cdot N$ clicks to make sure that GBF has been stable. Figure 2(a) shows the theoretical and experimental results. We make the observation that the experimental result of GBF algorithm is very close to the theoretical result when detecting duplicates over jumping windows. When $k = 10$ and $m = 1,876,246$, the false positive rate is only about 0.007.

When evaluating our TBF algorithm over sliding windows, we set the sliding window size to $N = 2^{20}$. We generated $20 \cdot N$ distinct click identifiers, and counted the false positives within the last $10 \cdot N$ clicks to make sure that TBF has been stable. Figure 2(b) shows the theoretical and experimental results. We make the observation that the experimental result of TBF algorithm is very close to the theoretical result when detecting duplicate clicks over sliding windows. When $k = 10$ and $m = 15,112,980$, the false positive rate is only about 0.001.

6 Conclusions

In this paper, we address the problem of detecting duplicate clicks in pay-per-click streams over jumping windows and sliding windows. We propose group Bloom filters which significantly reduces the memory operations when processing click streams, and our GBF algorithm based on group Bloom filters is effective and efficient over jumping windows. To detect duplicate clicks over sliding windows and jumping windows with large number of sub-windows, we propose an innovative TBF algorithm based on a new data structure called timing Bloom filter, which can process click streams over sliding windows using less memory space and processing time. Both theoretical analysis and experimental results show that our algorithms are effective and efficient in terms of false negative rate, false positive rate and running time when detecting duplicate clicks.

In the future, we will continue to explore the issues of click fraud and click quality under data stream models. We will consider various sophisticated click fraud attacks, and study advertising network dynamics, new service models, economic and social impacts of click frauds.

7 Acknowledgments

We thank the anonymous reviewers for their invaluable feedback. This work was partially supported by NSF under grants No. CNS-0644238, CNS-0626822, and DUE-0313837, ARDA under contract number NBCHC030107, and Carver Trust Foundation.

References

[1] Google click fraud settlement. http://googleblog.blogspot.com/pdf/lanes_google_final_order.pdf, Mar. 2006.

[2] Yahoo click fraud settlement. http://yhoo.client.shareholder.com/ReleaseDetail.cfm?ReleaseID=202354, June 2006.

[3] V. Anupam, A. Mayer, K. Nissim, B. Pinkas, and M. K. Reiter. On the security of pay-per-click and other web advertising schemes. In *WWW 1999*.

[4] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *PODS 2004*.

[5] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS 2002*.

[6] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *PODS 2003*.

[7] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[8] A. Broder and M. Mitzenmacher. Network applications of Bloom filters: A survey. In *the 40th Annual Allerton Conference on Communication, Control, and Computing*.

[9] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *SODA 2002*.

[10] F. Deng and D. Rafiei. Approximately detecting duplicates for streaming data using stable bloom filters. In *SIGMOD 2006*.

[11] eMarketer. Online ad spending to total $19.5 billion in 2007. http://www.emarketer.com/Article.aspx?1004635, Feb. 2007.

[12] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *Transactions on Networking*, 8(3):281–293, 2000.

[13] M. Gandhi, M. Jakobsson, and J. Ratkiewicz. Badvertisements: Stealthy click-fraud with unwitting accessories. *Anti-Phishing and Online Fraud, Part I Journal of Digital Forensic Practice*, 1, Nov. 2006.

[14] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2002.

[15] P. B. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *SPAA 2002*.

[16] L. Golab, D. DeHaan, E. D. Demaine, A. López-Ortiz, and J. I. Munro. Identifying frequent items in sliding windows over on-line packet streams. In *IMC 2003*.

[17] Internet Advertising Bureau of United Kingdom. IAB online adspend study - 2006. http://www.iabuk.net/en/1/iabknowledgebankadspend.html.

[18] L. Lee and H. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *PODS 2006*.

[19] X. Lin, H. Lu, J. Xu, and J. X. Yu. Continuously maintaining quantile summaries of the most recent N elements over a data stream. In *ICDE 2004*.

[20] A. Metwally, D. Agrawal, and A. E. Abbadi. DETECTIVES: DETEcting Coalition hiT Inflation attacks in adVertising nEtworks Streams. In *WWW 2007*.

[21] A. Metwally, D. Agrawal, and A. E. Abbadi. Duplicate detection in click streams. In *WWW 2005*.

[22] A. Metwally, D. Agrawal, and A. E. Abbadi. Using association rules for fraud detection in web advertising networks. In *VLDB 2005*.

[23] A. Metwally, D. Agrawal, A. E. Abbadi, and Q. Zheng. On hit inflation techniques and detection in streams of web advertising networks. In *ICDCS 2007*.

[24] S. Muthukrishnan. Data streams: Algorithms and applications. Technical report, Rutgers University, 2003.

[25] Outsell Survey. Hot topics: Click fraud reaches $1.3 billion, dictates end of "don't ask, don't tell" era. http://www.outsellinc.com/store/products/243.

[26] M. K. Reiter, V. Anupam, and A. Mayer. Detecting hit shaving in click-through payment schemes. In *Proceedings of the 3rd USENIX Workshop on Electronic Commerce*, 1998.

[27] S. Tirthapura, B. Xu, and C. Busch. Sketching asynchronous streams over sliding windows. In *PODC 2006*.

[28] L. Zhang and Y. Guan. Frequency estimation over sliding windows. In *ICDE 2008*.

[29] L. Zhang and Y. Guan. Variance estimation over sliding windows. In *PODS 2007*.

[30] L. Zhang and Y. Guan. Detecting click fraud in pay-per-click streams of online advertising networks. Technical report, Iowa State University, Mar. 2008.

[31] Y. Zhu and D. Shasha. StatStream: Statistical monitoring of thousands of data streams in real time. In *VLDB 2002*.