A New Memory Efficient Technique for Fraud Detection in Web Advertising Networks

Howida A. shedeed
dr_howida@cis.asu.edu.eg
Faculty of Computer and Information Sciences
Ain Shams University, Cairo, Egypt

Summary
The advertising network considered as the middle man in web advertising between advertisers and publishers. This paper presented an intelligent and memory efficient Fraud detection technique with intelligent classification engine to be used by the advertising networks to scan clicks and impressions offline streams happen on publisher side for the purpose of detecting click fraud and impression fraud. The proposed classification technique is based on the proposed data structure for a Scalable Dynamic Counting Bloom Filter (SDCBF). It is a hybrid structure between the Scalable Bloom Filter (SBF) and the Counting Bloom Filter (CBF). It is a variant of the CBF in such a way that, the counter is a dynamic size bit array that can adapt dynamically to its content. Both theoretical analysis and experimental results show that, the investigated technique can achieve minimum space storage with low false positive rate when detecting both duplicate clicks over a sliding window and fast click.

Key words:
Fraud detection – web applications – Advertising Network.

1. Introduction

Internet advertising is a major and necessary component of the competitive marketing strategy of most companies. Spread of fraud by unscrupulous advertiser's competitors and publishers will cause severe damage to the advertiser and the advertising network. Figure1. Shows the click traffic model in the advertising network.

Since publishers are paid by the traffic they drive to the advertisers, 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 advertising commissioners (advertising networks) and sometimes in extra costs or paying reimbursements for advertisers. So advertising networks are in need to detect the fraud happen on traffic derived by the publisher web site accurately and securely.

One of the main challenges involved in detecting such fraudulent behavior is maintaining the privacy of the web users, furthermore, the challenging scale of the click and impression streams in an ever growing Internet [9].

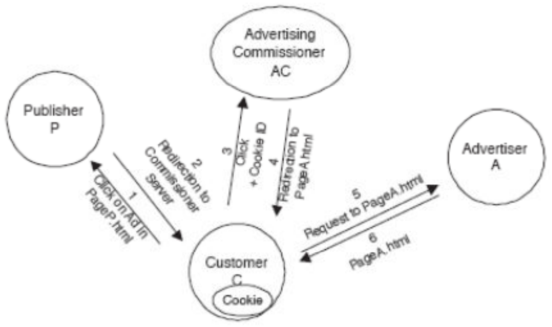

Advertising Commissioner AC
Publisher P
Advertiser A
Customer C
Cookie
1 Click on Ad in PageP.html
2 Redirection to Commissioner Server
3 Click + Cookie ID
4 Redirection to PageA.html
5 Request to PageA.html
6 PageA.html

Figure1. Clicks Traffic in Advertising Networks.

In this research we develop a professional and memory efficient tool for assisting advertising networks in scanning and analyzing click and impression streams to detect any potential fraud from the dishonest publisher or the dishonest advertiser competitors. The system applied two detection techniques: the duplicate detection technique which applied for detecting duplicates from both click and impression streams within a user defined time span (window). It considers a click or impression to be duplicate if it exceeds a user defined threshold of legal number of duplicates within this span [6]. The second technique is used for detecting non-human behavior (robotic traffic) which known as fast click detection. In this technique we link the clicks with impressions together to detect the clicks that happen without impressions (automated clicks) or after a short period of time less than a user defined threshold [2].

We develop in this research an intelligent classifier engine to be used with the two techniques for fraud detection. The proposed engine design is based on the proposed probabilistic data structure for a Scalable Dynamic Counting Bloom Filter (SDCBF) which provides space efficient storage for sets with minimum probability of false positives on membership queries. SDCBF is a hybrid structure between the scalable bloom filter and the

Manuscript received August 5, 2010
Manuscript revised August 20, 2010

counting bloom filter with enhancement added to its structure to fit out our memory efficient solution. It is a variant from the counting bloom filter in such a way that, the counter is a dynamic size bit array that can adapt dynamically to the count number.

Sec 2 presented the previous work in this research area. Sec 3 presented the proposed structure for the scalable dynamic counting Bloom filter SDCBF. Sec 4 illustrate the application of the fraud detection techniques using the proposed engine design. Sec 5 shows the system design and the results of the experiment. Finally conclusion and future work are drawn in Sec 6.

2. Previous Work

Many algorithms has been investigated in this application such as [1] which proposed a simple solution to detect duplicate clicks using a bit vector. The algorithm keeps track of which elements have been observed in the stream by flagging their corresponding bits in the bit vector to 1. A new element is a duplicate if its bit has been flagged to 1 in the Bit Vector. The algorithm is simple, exact, and takes O (1) steps and space to insert a new element into the bit vector, or to check it for duplication. However, this simple scheme cannot be implemented in our case. The alphabet we are dealing with in this application is the domain of IDs of clicks which are represented by 64 characters. Thus, keeping a bit for every ID entails keeping 2^{512} bits ≈ $1.676 * 10^{153}$ bytes, which is infeasible.

[1] Proposed an improvement of the above solution called "The overlapping bit-substrings solution". The algorithm keeps partial information, rather than all the combinations of the alphabet b = 64 Character = 512 bits. It keeps less than 2^b bits, but still get approximate results with a very low error (false positive). [1] Modified his algorithm once more to serve both purposes of achieving better results, and facilitating the probabilistic analysis. He used the same idea of shrinking the size of the bit vector to less than 2b. However, instead of using overlapping bit-substrings of the IDs, he used an independent hash functions. Interestingly, using independent hash functions makes his solution another development of Bloom Filters [5].

2.1 Bloom Filter

A Bloom Filter [5] is a probabilistic data structure that was proposed to detect approximate membership of elements.

Given two sets, X, and Y, the Bloom Filter algorithm would loop on every element in set X, to check if it belongs to set Y, too. The algorithm is probabilistic, requires $O(|X|)$ operations, $O(|Y|)$ space, and d independent hash functions. A Bloom Filter can assert that an element in *X* does not belong to *Y*, but cannot assert that an element in *X* belongs to *Y*. That is, its errors are only false positive, and never false negative.

An empty Bloom Filter as shown in figure 2 is an array of M cells that are initially zeroed. Each element, y, in Y is hashed using d independent hash functions to addresses $y_1, y_2, \ldots, y_d$, which are set to 1, such that $0 \le y_i \le M - 1, \forall i$. For each element, x, in X, its d hash results, x_1 to x_d, are generated in the same manner, and checked against the Bloom Filter that represents the set Y. If any of the cells x_1 to x_d is not set to 1, then it can be asserted that $x \notin Y$. If all the cells x_1 to x_d are set to 1, then there is a good probability that $x \in Y$. The interesting thing about Bloom Filters is that they do not store the elements of the set whose membership is tested. This is very useful in cases where the IDs of the elements are huge, like in our case.

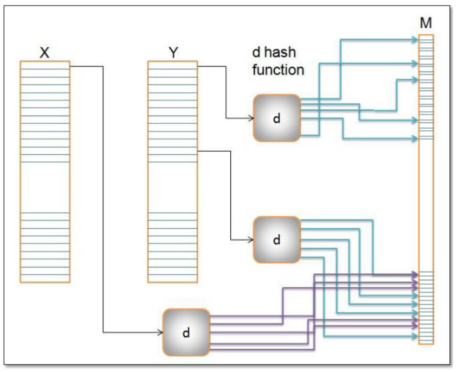


Figure 2. Bloom Filter with d=5 hash functions

The probability of a false positive is inversely proportional to *d*, the number of hash functions used, given that the space utilized grows proportionally with *d*.

[1] used M bit array for his Bloom Filter where *M* is *O*(*N*), and *N* is the estimated size of the processed window.

The above algorithm for using Bloom Filter has some disadvantages such as: It has a fixed space size according to false positive error, so it is impossible to store extra elements without increasing the false positive probability. It does not count the number of duplication for each element and finally it is not scalable for more elements.

2.2 Stable Bloom filter

[6] Proposed Stable Bloom filters as a variant of Bloom filters for streaming data. The idea is that since there is no way to store the entire history of a stream (which can be infinite). Stable Bloom continuously evicts the stale information to make room for those more recent elements.

The authors show that a tight upper bound of false positive rates is guaranteed, and the method is superior in terms of both accuracy and time efficiency when a small space and an acceptable false positive rate are given.

2.3 Counting Bloom filter

[1] Proposed a modification of Bloom Filters to enable Bloom Filters to count the duplicates. The underlying idea is to replace the array of M bits with an array of M counters, thus we have a Counting Bloom Filter (CBF). The same idea was proposed by [7] to modify Bloom Filters for implementing a scalable distributed cache sharing protocol.

For every element that is to be inserted, increment the d counters to which the element hashes. To delete an element, decrement the *d* counters to which the element hashes. The number of hash functions *d* can be determined according to the required error rate as we will illustrate later.

The disadvantages of this technique, which is avoided in the proposed technique in this paper, is that the array positions for Bloom filter are extended from being a single bit, to an n-bit counter. The size of counters is usually 3 or 4 bits. Hence counting Bloom filters use 3 to 4 times more space than static Bloom filters. With huge number of membership for our application this will be inefficient in using space, and the CBF size will enlarge to uncontrollable size. Also we may suffer from an overflow problem when the counter reached 2^4 for a 4 bit counter for example.

Many applications used the CBF such as [8] which used the CBF for network intrusion detection and [11] which proposed a comprehensive service-storage solution using the CBF.

2.4 Cache Counting Bloom filter

[4] Introduced a multi-level memory hierarchy and a special hardware cache architecture for counting Bloom filters that is utilized by network processors and packet processing applications such as packet classification and distributed web caching systems. Based on the value of the counters in the counting Bloom filter, a multi-level cache architecture called the cache counting Bloom filter (CCBF) is presented and analyzed. The results show that the proposed cache architecture decreases the number of memory accesses (but not the memory size) by at least 51.3% when compared to a standard Bloom filter.

2.5 Scalable Bloom Filter

Scalable Bloom Filter [10] provides a solution for the case in which not only is the number of elements not known in advance but also we need to strictly enforce some maximum error probability.

3. Proposed Technique

In this paper the author presents a new technique for fraud detection based on the author’s investigated structure for the Bloom filter called a Scalable Dynamic Counting bloom filter (SDCBF). The proposed structure for SDCBF is a hybrid structure of the (SBF) introduced in [8] and the (CBF) introduced in [1] to benefit from the advantages of both architectures. But its variant from the CBF structure in such a way that, the filter is a linked list of bit arrays and each bit array is dynamic in size according to the stored number. Initially the counter is null until its index is referenced, then the engine replaces the counter with a bit array with initial user defined size. The system enlarges or reduces the array size according to the new stored number in the counter after incrementing or decrementing the counter. Each counter is independent of the other counters. Both theoretical analysis and experimental results show that our proposed software architecture for the SDCBF can achieve minimum space storage with at least 50% less than the structure used for the traditional CBF that uses a constant counter size. Another advantage in the proposed structure is that, it is impossible to suffer from overflow problem when incrementing the counter, since the counter’s size can dynamically increased to accommodate the new number.

The investigated SDCBF as shown in figure 3 consists of a list of slices. Each slice consists of a linked list of counters and each counter is a dynamic bit array counter.

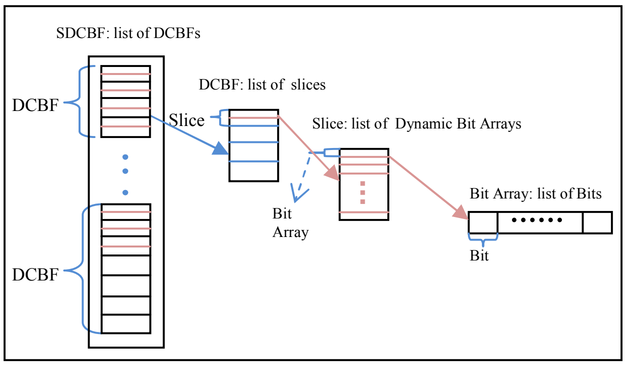


Figure 3 The Proposed Architecture for the SDCBF

Thus the SDCBF is made up of a series of one or more Dynamic Counting Bloom Filter. Querying is made by testing for the presence in each filter. Each successive filter is created with a tighter maximum error probability on a geometric progression, so that the compounded probability over the whole series converges to some wanted value, even accounting for an infinite series [10].

In the classical Bloom Filter all hash functions are used to generate indexes over M. Since these hash functions are independent, nothing prevents collisions in the outputs. In the most extreme case we could have $h_1(x) = h_2(x) = . . . = h_k(x)$ for k hash functions. This means that in the general case each element will be described by 1 to k distinct indexes.

Although for large values of M a collision seldom occurs, this aspect makes some elements more prone to false positives. So we partitioning the M array among the k hash functions, thus creating k slices of $m = M/k$ bits, In this variant, each hash function $h_i(x)$, with $1 \le i \le k$, produces an index over m for its respective slice.

The SDCBF starts with one filter with k_0 slices and error probability P_0 (False Positive), number of slices= number of hash functions $k_0 = \log_2(1/P)$. When this filter gets full according to the tightening ratio r, a new one is added with k_1 slices and error probability $P_1 = P_0 * r$, where $0 < r < 1$. At a given moment we will have l filters with error probabilities P_0, P_0r, P_0r^2,… P_0r^{l-1}. The compounded error probability for the SBF will be $P = 1 - \prod_{i=0}^{l-1}(1 - p_0 r^i)$. The number of slices for the next counting bloom filter $k_i = k_0 + i * \log_2 r^{-1}$.

SDCBF able to adapt to variations in size of several orders of magnitude in an efficient way. When a new filter is added to a SDCBF, its size can be chosen orthogonally to the required false positive probability. A flexible growth can be obtained by making the filter size grows exponentially. We can have a SDCBF made up of a series of filters with slices having sizes m_0, m_0s, m_0s^2, . . . ,m_0s^{l-1}. Considering the choice of s = 2 for small expected growth and s = 4 for larger growth according to the server memory size.

4. **Applying The Detection Techniques Using The Engine**

This section illustrate how we can use the proposed structure for the Scalable Dynamic Counting Bloom Filter on applying the two fraud detection techniques, the duplicate detection technique and the fast click detection technique.

4.1 The Duplicate Detection Technique

In this solution the investigated structure for the SDCBF for counting the number of duplicate is used for both impression stream and click stream. The implemented algorithm uses 5 independent hash functions.

Initially the advertising network has to define a threshold for the allowable number of duplicates. If the number of duplicates for certain ID in the counting Bloom filter exceeds this threshold then this ID will be classified as fraud element and then added to a data dictionary that we used for collecting the indexes of the duplicated elements and their numbers of duplication. If this ID already exist in the dictionary then update its value, else add new key value pair in the dictionary. The duplicate detection technique is applied using sliding window approach with a specified time span defined by the advertising commissioner.

Slide the window for every new entry arriving with its corresponding time span. If the new index for the bottom border of the window is greater than or equal to the stream size then terminates and display the output dictionary, else this mean that there are one or more new elements. So repeat the previous task with the new sliding window.

4.2 Fast Click Detection

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.

Assuming that, the duplicate impressions and clicks are already purged, therefore, we will not face the problem of finding more than one impression matching one click. The technique 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_I), it is required to check if the last impression with the same ID_I of the click has a $timestamp_I$, such that ($timestamp_J$ − $timestamp_I$) $< \tau$.

An approximate solution introduced by [2] and also used in [3] is as follows: The technique 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, γ, 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 $\gamma + 1$ 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 γ 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. [2] Suggested to use a vector of integers with independent hash functions like the one he used for duplicates which is a memory consuming solution. In this research, the author suggested to use a list of dynamic bit arrays with independent hash functions like the one used for detecting duplicates, which is a memory efficient structure. Searching in this list of bit arrays entails hashing the clickID using the independent hash functions and checking if all the corresponding bit arrays are non-zero. Inserting an element entails hashing the impression ID using the independent hash functions, and incrementing all the corresponding bit arrays. Deleting an element entails hashing the impression ID using the independent hash functions, and decrementing all the corresponding bit arrays. A bit array 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 γ 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.

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 bit arrays is 0 in New-Imps, and all the corresponding bit arrays are non-zero in Old-Imps.

5. System Design and Experimental Results

5.1 The Generic Clustered Web Server Architectural Model

This section explains how the fraud detection system can be deployed in a real clustered Web server architecture. Figure 4. Shows a generic web server architecture model borrowed from [12]. As shown in the figure, 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. Two consecutive requests from the same site, and the 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.

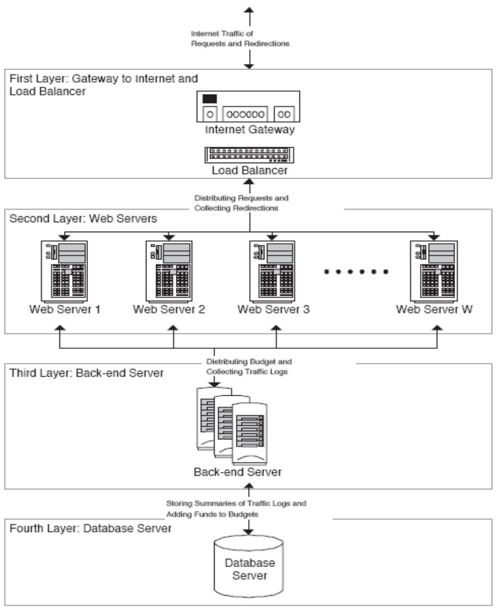

Internet Traffic of Requests and Redirections
First Layer: Gateway to Internet and Load Balancer
Internet Gateway
Load Balancer
Distributing Requests and Collecting Redirections
Second Layer: Web Servers
Web Server 1
Web Server 2
Web Server 3
Web Server W
Distributing Budget and Collecting Traffic Logs
Third Layer: Back-end Server
Back-end Server
Storing Summaries of Traffic Logs and Adding Funds to Budgets
Fourth Layer: Database Server
Database Server

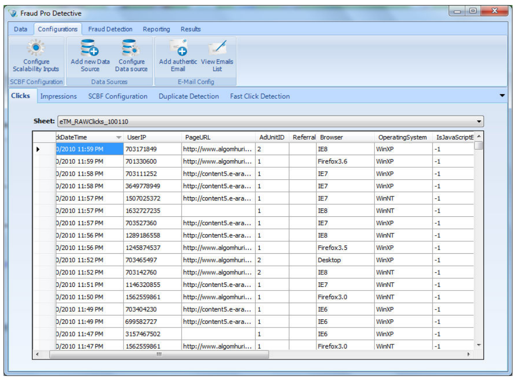
Figure 4. The Advertising Networks' Generic Web Server Architectural Model.

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.

5.2 System Interface and Experimental Results

To illustrate and prove the idea of this research a simple prototype system has been implemented to do the experiment and explain the results.

Figure 5. Shows the main user interface screen for the system and also explains the data source after reading.



Sheet: eTM_RAWClicks_100110

ːkDateTime	UserIP	PageURL	AdUnitID	Referral	Browser	OperatingSystem	IsJavaScriptE
0/2010 11:59 PM	703171849	http://www.algomhuri...	2		IE8	WinXP	-1
0/2010 11:59 PM	701330600	http://www.algomhuri...	1		Firefox3.6	WinXP	-1
0/2010 11:58 PM	703111252	http://content5.e-ara...	1		IE7	WinXP	-1
0/2010 11:58 PM	3649778949	http://content5.e-ara...	1		IE7	WinXP	-1
0/2010 11:57 PM	1507025372	http://content5.e-ara...	1		IE7	WinNT	-1
0/2010 11:57 PM	1632727235		1		IE8	WinNT	-1
0/2010 11:57 PM	703527360	http://content5.e-ara...	1		IE7	WinXP	-1
0/2010 11:56 PM	1289186558	http://content5.e-ara...	1		IE8	WinNT	-1
0/2010 11:56 PM	1245874537	http://www.algomhuri...	1		Firefox3.5	WinXP	-1
0/2010 11:52 PM	703465497	http://www.algomhuri...	2		Desktop	WinXP	-1
0/2010 11:52 PM	703142760	http://www.algomhuri...	2		IE8	WinNT	-1
0/2010 11:51 PM	1146320855	http://content5.e-ara...	1		IE7	WinNT	-1
0/2010 11:50 PM	1562559861	http://www.algomhuri...	1		Firefox3.0	WinNT	-1
0/2010 11:49 PM	703404230	http://content5.e-ara...	1		IE6	WinXP	-1
0/2010 11:49 PM	699582727	http://content5.e-ara...	1		IE6	WinXP	-1
0/2010 11:47 PM	3157467502		1		IE6	WinXP	-1
0/2010 11:47 PM	1562559861	http://www.algomhuri...	1		Firefox3.0	WinNT	-1

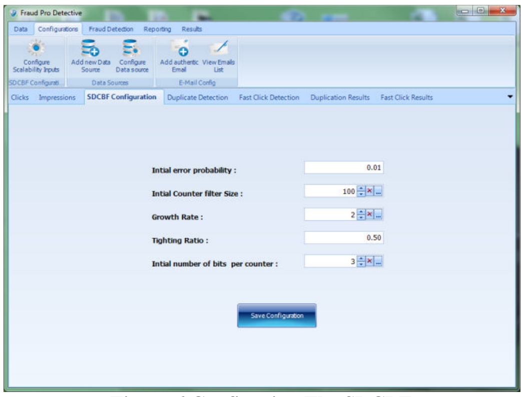
Figure 5 the data source after reading

Figure 6 is the interface screen used to configure the SDCBF, the "Initial counter filter size" defines the maximum expected number of duplicates initialized by the Ad. network. Also the Ad. network's user defines the "Initial number of bits per counter" which defines the initial size for the dynamic bit array counter. Afterword according to the proposed technique the counter's size will increased or decreased to accommodate the new stored number after incrementing or decrementing the counter.

Figure7 Used to configure the classification engine for duplicate detection by specifying the threshold for the allowable number of duplicates and the "minimum delay time" which defines the sliding window size. The system also enables the user to implement the algorithm on a subset of the stream not the all stream.

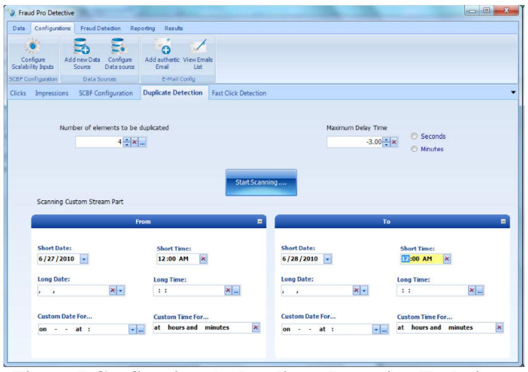
Figure 8 explains the interface screen used to configure the fast click detection engine by defining τ: the minimum allowable delay between impression and click and γ: the scan time period threshold.

Finally Figure 9 shows the output report for 'duplicate detection" experiment after analyzing the input stream. The experiment is done with maximum allowable number of duplicate = 4 and the initial counter size= 3bit, the result shows that for total count of 2039, 98 duplicates are found from 38 users. The number of duplicates found is 2 to 9 which required 2 to 4 bits counter. Only 22 of 98 duplicates are exceeds 7, which required 4 bit counter and the remaining 77 required only 3 bits counters. The remaining 1941 count required only maximum 2 bits counters. Thus, the total size for the required filter will be at maximum 22*4+77*3+1941*2= 4023 bit. But in all the preceding research that uses a fixed size CBF, the required CBF's size=2039*4=8156 in this case. Thus the proposed structure achieved minimum space storage with at least 51% less than the structure used for the traditional CBF such as the one used in [1], [7] and [8], that uses a constant counter size.



Intial error probability : 0.01
Intial Counter filter Size : 100
Growth Rate : 2
Tighting Ratio : 0.50
Intial number of bits per counter : 3

Figure 6 Configuring The SDCBF

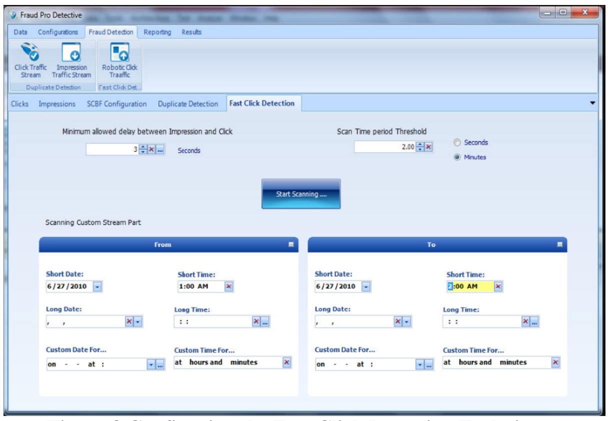


Number of elements to be duplicated: 4
Maximum Delay Time: -3.00 Seconds Minutes

Scanning Custom Stream Part

From		To	
Short Date: 6/27/2010	Short Time: 12:00 AM	Short Date: 6/28/2010	Short Time: 12:00 AM
Long Date: , ,	Long Time: : :	Long Date: , ,	Long Time: : :
Custom Date For...: on - - at :	Custom Time For...: at hours and minutes	Custom Date For...: on - - at :	Custom Time For...: at hours and minutes

Figure 7 Configuring the Duplicate Detection Technique



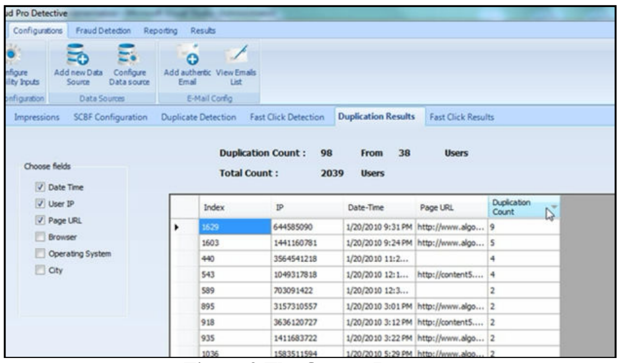
Minimum allowed delay between Impression and Click: 3 Seconds

Scan Time period Threshold: 2.00 ( ) Seconds [✓ Minutes]

Scanning Custom Stream Part

From		To	
Short Date: 6/27/2010	Short Time: 1:00 AM	Short Date: 6/27/2010	Short Time: 2:00 AM
Long Date: , ,	Long Time: : :	Long Date: , ,	Long Time: : :
Custom Date For... on - - at :	Custom Time For... at hours and minutes	Custom Date For... on - - at :	Custom Time For... at hours and minutes

Figure 8 Configuring the Fast Click Detection Technique.



Duplication Count : 98 From 38 Users
Total Count : 2039 Users

Choose fields
[✓] Date Time
[✓] User IP
[✓] Page URL
[ ] Browser
[ ] Operating System
[ ] City

Index	IP	Date-Time	Page URL	Duplication Count
1629	644585090	1/20/2010 9:31 PM	http://www.algo...	9
1603	1441160781	1/20/2010 9:24 PM	http://www.algo...	5
440	3564541218	1/20/2010 11:2...		4
543	1049317818	1/20/2010 12:1...	http://content5....	4
589	703091422	1/20/2010 12:3...		2
895	3157310557	1/20/2010 3:01 PM	http://www.algo...	2
918	3636120727	1/20/2010 3:12 PM	http://content5....	2
935	1411683722	1/20/2010 3:22 PM	http://www.algo...	2
1036	1583511594	1/20/2010 5:29 PM	http://www.algo...	2

Figure 9 The Output Report

6. Conclusion and Future Work

This paper presented the architecture and design of an offline stream scanning tool with intelligent classification engine that scans click and impression streams happen on publisher side for the purpose of detecting click fraud and fast click fraud. The proposed engine design is based on the proposed probabilistic data structure for the Scalable Dynamic Counting Bloom Filter (SDCBF). SDCBF is used in the two techniques for Fast Click Detection and Duplicate Detection over Impression and Click streams.

This paper introduced the Scalable Dynamic Counting Bloom Filters (SDCBF), a mechanism that allows representing sets without having to know the maximum set size and yet being able to choose from the start the maximum false positive probability. The mechanism adapts to set growth by using a series of classic Dynamic Counting Bloom Filters of increasing sizes and tighter error probabilities, added as needed. Also this mechanism is efficient in memory usage for the counters in such a way that, each counter is a dynamic bit array that can adapt in size to the stored number.

In Duplicate Detection Technique, a sliding window is applied on the stream to detect duplication given the threshold for the allowable number of duplicates and the window's time span.

In Fast Click Detection technique, a sliding window is applied on the click stream and store its corresponding impressions in two Scalable Dynamic Counting Bloom Filter; New-Imp and Old-Imp which classified according to τ, the minimum allowable delay between impression and click, and γ, the scan time period threshold, respectively.

Future work will include: Applying the SDCBF in Online monitoring and also trying to propose an efficient techniques to detect the publishers' coalition fraud and the hit Shaving attacks.

Acknowledgments

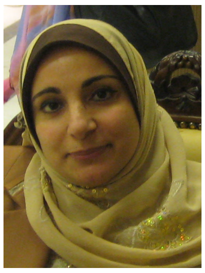
I want to acknowledge A. E. Mohamed and H. M. Saad for helping me in coding the implementation to prove the idea of this research.

References

[1] A. Metwally, D. Agrawal, and A. El Abbadi, "Duplicate Detection in Click Streams", In Proceedings of the 14th International conference on World Wide Web, ,May 10-14, 2005, Chiba, Japan
[2] A. Metwally, D. Agrawal, and A. El Abbadi. "On Hit Inflation Techniques and Detection in Streams of Web Advertising Networks.", 27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007), June 25-29, 2007, Toronto, Ontario, Canada. IEEE Computer Society 2007.
[3] A. Metwally, D. Agrawal, A. El Abbadi, "Detectives: detecting coalition hit inflation attacks in advertising networks streams", In Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007.
[4] Ahmadi, Mahmood; Wong, Stephan, "A Cache Architecture for Counting Bloom Filters", 15th international Conference on Netwroks (ICON-2007).
[5] Burton H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors", Communications of the ACM, 13(7):422–426, 1970.
[6] Fan Deng and Davood Rafiei, "On Approximately detecting duplicates for streaming data using stable bloom filters", In Proceedings of the 2006 ACM ,SIGMOD International Conference on Management of data, Pages: 25 - 36, 2006.
[7] Fan Li Cao, Pei Almeida, Jussara Broder, Andrei (2000), "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol", IEEE/ACM Transactions on Networking **8** (3): 281–293.

[8] Harwayne Gidansky J., Stefan D., Dalal I., “FPGA-based SoC for real-time network intrusion detection using counting bloom filters”, In proceeding of the IEEE SoutheastCon 2009.
[9] Markus Jakobsson and Zulfikar Ramzan, “Crimeware”, reference book, Symantec Press ,2008.
[10] Paulo Sérgio Almeida, Carlos Baquero, Nuno Preguica, Hutchison, “Scalable Bloom Filter”, Information Processing Letters, Pages: 255-261, March 2007.
[11] Shuxing Cheng Chang, C.K. Liang-Jie Zhang, “An Efficient Service Discovery Algorithm for Counting Bloom Filter-Based Service Registry”,. IEEE International Conference on Web Services, July 2009. ICWS 2009.
[12] 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.

Author:



Howida AbdelFattah Saber Shedeed received the PHD. degree in electrical engineering, Computers and systems engineering, from Ain Shams university, faculty of engineering 2005. She is an Assistant professor in Scientific Computing department, faculty of computers and information sciences, Ain Shams University, Cairo, Egypt.