# Designing a Click Fraud Detection Algorithm
## *Exposing Suspect Networks*

Dimitris Antoniou[1], Christos Makris[1], Despina Meridou[1],Giannis Tzimas[2] and Emmanouil Viennas[1]
[1]*Department of Computer Engineering and Informatics, University of Patras, Patras, Greece*
[2]*Department of Applied Informatics in Management & Economy, Faculty of Management and Economics, Technological Educational Institute of Messolonghi, Messolonghi, Greece*

Keywords: Burst Web Visits, Click Fraud, Burst Tries.

Abstract: As the use of the Web expands and appears almost everywhere in business' practices, new algorithmic problems appear and need to be efficiently handled; one of them that has attracted the attention of both researchers and practitioners is click fraud. Click fraud can be defined as the practice of repetitively clicking on search ads without being actually interested in the content of the related links, with the intention of either increasing the Website's profits or exhausting an advertiser's budget. In this work, we propose an algorithm, which exposes suspect networks instead of single IPs, based on utilizing efficient data structures that have not been employed in previous works.



## 1 INTRODUCTION

The emerging size of the World Wide Web, and its growing pattern of use, creates new challenges and new algorithmic problems that need to be efficiently handled. One of these problems that has attracted the attention of researchers is click fraud detection. As going online becomes common practice to more and more commercial enterprises, it has become vital for them to be advertised online to customers. An online ad is usually placed in the Web page of a hosting Website as a banner and the owner of the Web site gets paid a fee for each click on the ad from a casual customer; this is the pay-per-click model (PPC) of online advertisement, sometimes also called cost-per-click model. Click fraud is a problem that is deeply connected with this model of online advertisement and is intimately connected (or, better, related) to the problem of burst of visits (Zhou et al., 2004), affecting a specific Web page.

Click fraud can be formally defined as the practice of deliberately and repetitively clicking on search ads with the intention of either having third-party Website profits increase, or of depleting the budget that the advertisers have charged for this click (Sakkopoulos et al., 2010). In both cases, instead of real users automated scripts or programs can simulate multiple clicks on the ad, and since the advertiser gets charged without taking any true values, the term "fraud" is naturally used. Other similar terms are "unwanted clicks" or "invalid clicks"; this terminology is sometimes used by Google since the word "fraud" can have legal implications or be difficult to be proved. It should be noted that Google publishes reports concerning the dangers behind click fraud and defines them as clicks produced by methods that are normally prohibited.

Since this deliberate action of misleading an advertiser can be naturally justified only if the gains are large, the number of clicks need to be quite large for someone to suspect that something is going wrong, hence bursts of visits to the specific site accompanied with clicks on the related ad may help to indicate the probable presence of a click fraud. A pattern of visits or accesses is considered *bursty* when it occurs with high intensity over a limited period of time and such patterns have been also observed in various Web applications in a number of studies (Zhou et al., 2004), since they can be utilized for personalization purposes and as indicators of other interesting events. A burst scenario can be algorithmically characterized and captured by using counters and recording for each Web page the number of accesses in a fixed time interval. The basic principles behind detecting click fraud exploiting burst of visits is to employ thresholds in the frequency of visits from the various IP addresses

Antoniou D., Makris C., Meridou D., Tzimas G. and Viennas E..
Designing a Click Fraud Detection Algorithm - Exposing Suspect Networks.
DOI: 10.5220/0004369300930098
In *Proceedings of the 9th International Conference on Web Information Systems and Technologies* (WEBIST-2013), pages 93-98
ISBN: 978-989-8565-54-9

and locate as responsible the IP behind the majority of the visits.

In this work we aim at detecting click fraud real-time and ban the networks responsible, by presenting the design of an innovative solution which considerably increases the security in such cases by detecting bursts caused by networks. The novelty in our approach lies in the fact that our technique locates not individual IPs but *whole sub-networks*, since it is able to detect the common prefix (that is the sub-network) in the IP addresses that are responsible for bursts of visits and depict similar in frequency patterns of accesses under a dominant one; this was not possible in previous techniques that focused on individual IP addresses.

The rest of the paper is organized as follows. In section 2, the related work is presented. In section 3, our proposed technique in order to efficiently detect networks that are suspects for click fraud is described. Finally, conclusions and future work are discussed in section 4.

## 2 RELATED WORK

There are various techniques that can be employed in order to detect click fraud, which are mainly based on the application of data mining and machine learning techniques in order to provide detailed evidence of possible click fraud patterns. The work of (Antoniou et al., 2011) is the most resembling to ours in the problem. In this paper, an approach for click fraud detection was proposed, that analyzed the number of visits during a certain time interval, depending on the kind of the investigated click fraud and on the application at hand (e.g. poll, posts).

The novelty of that approach was the effective use of novel data structures for reducing the space consumption and allowing the real time detection of click fraud. However, that work was not capable of dealing with bursts of visits for which not just an IP address, but a whole network is responsible. In this case, multiple IPs, belonging to a greater network, perform visits to the banner on a Web site. Since every IP is unique, the algorithm posed by the authors cannot detect a single suspect IP and the click fraud event remains undetected. In order to detect such kinds of click fraud, there should exist a data structure, which would be able to return not the most recent and frequent IP address, but an equivalent prefix. This prefix would represent the network, to which, a group of IPs responsible for the burst of visits belongs. To do so, we utilize another advanced data structure, the burst tries (Heinz et al., 2002); as we will describe in the sequel, burst tries have interesting properties that permit the efficient handling and detection of such kind of activities.

Before proceeding to the main section, we should mention that, in this paper, the term “burst” is used to describe two different concepts: (1) the bursting process in a burst trie, which will be described later and (2) a “burst” of visits: the case where few Web pages become very popular for short periods of time and are accessed very frequently in a limited temporal space.

In order to distinguish between these two different concepts, we will hereafter refer to a “burst” of visits using quotes.

## 3 CLICK FRAUD DETECTION USING BURST TRIES

A burst trie is an elegant data structure tailored to store strings, that is series of symbols taken from a given alphabet, and it can be considered to be a hybrid data structure composed of two levels; the upper level resembling a traditional trie storing the strings according to the alphabetic representation of prefixes of them with a specific length and the lower level that stores the remaining parts of the strings using a dictionary data structure chosen according to the specific application at hand. In particular, the upper level of the structure is termed as *Access trie* and it resembles a traditional trie with the difference that the leaves, instead of atomic elements, store sets of elements. Each of these sets of elements is stored to the so called *Containers* that are implemented with data structures suitable to store small sets of elements. The structure begins with an empty container and is equipped with heuristics, three of which are described in (Heinz, et al., 2002), that are defined when a container is burst, and needs to be split in more than one containers. When a container at depth $i$ is burst, then it is replaced by a new trie node and a set of containers at depth $i+1$. From the above description, it is clear that for a container corresponding to a leaf of the Access trie with depth $i$, all its stored elements share a common prefix of length $i$, and this common prefix does not need to be stored in the data structure implementing the container. Burst tries have been shown to perform effectively in practice and in various applications are better than splay trees and ternary search trees that are natural competitors in various applications (Heinz et al., 2002).

The major property of the burst trie that suits our application is that if : (i) containers are implemented

using self-adjusting data structures, and (ii) the heuristic according to which a container is burst is based on how skew the accesses are in the elements of the container, then at any time instance the elements in the same container have similar access characteristics under a dominant one and can be represented by their common prefix, while the topmost element in the container is the one with the biggest share in the accesses.

In our algorithmic application and since we need to capture frequent patterns of accesses and hence represent strings where the frequency of access should be taken into account, we have chosen to implement the containers by employing splay trees.

## 3.1 Description of the Algorithm

In order to detect a "burst" of visits, we utilize a two dimensional matrix $B_{n\times m}$, with $n$ denoting the number of Web pages or data units and with $m$ representing the number of timestamps that are maintained for each data unit. We assign a number $w \in [0, m-1]$ to every Web page or data unit of the Website at random. Thus, each Web page or data unit corresponds to a row of the array $B_{n\times m}$. We define that, for the data unit $i$, a "burst" has occurred if and only if the difference between the timestamps stored in positions $(i, m-1)$ and $(i, 0)$ is equal or smaller than $T$, where $T$ is the period of time in which the visits should occur in order to suggest a "burst".

We should mention that in the algorithm described in (Antoniou et al., 2011) a splay tree was used in order to store Web pages and a stack of timestamps for each node of the splay tree. In our present approach, we simplify the employed data structures and we save time and space, since we have replaced the splay tree with a simple two dimensional array, while additionally by using the two dimensional array we are no longer in need of the stacks storing the timestamps of the various users' accesses.

In our algorithm, we use IP addresses represented in dot decimal notation and we represent every IP address as a string of bits. We store the visiting IPs by employing the burst trie data structure. In order to take into account the different Internet Protocols we utilize two different burst tries, depending on whether the visiting IP is defined as a 32-bit number (Internet Protocol Version 4 – IPv4) or as a 128-bit number (Internet Protocol Version 6 – IPv6). An Internet Protocol Version 4 (IPv4) address consists of 32 bits, which may be divided into four octets. These four octets are written in decimal numbers, ranging from 0 to 255, and are concatenated as a character string with full stop delimiters between each number. Similarly and as far as IPv6 is concerned, the 128 bits of the address are split in 16 octets and each two octets are represented with a hexadecimal four digits number.

As already mentioned, we use splay trees in order to implement the containers of the burst tries. Using the splaying technique, the most popular nodes of the tree, which represent parts of the visiting IP addresses, are rearranged so that they are located near the root of the container. Thus, the node that is being splayed to the root of the container could be quickly accessed in the future. For efficient splaying, a record in a splay tree requires three pointers, two pointing to its children and one to its parent, thus they use the most space of any of the container structures considered in (Sleator and Tarajn, 1985). However, it is a natural choice in our application since we want frequent IP accesses to be stored near the root of the container, in order to apply the burst heuristics that accompany the specific structure.

The general principle for maintaining a burst trie is to locate inefficient containers and burst them. In particular three heuristics are proposed in (Heinz et al., 2002); the *ratio*, the *limit* and the *trend* heuristics. In the *ratio* heuristic, a container is burst when the ratio of the number of accesses to the root of the container with the total number of accesses is less than a threshold, and simultaneously the number of accesses to the container is large enough. In the *limit* heuristic, the container is burst when the number of the elements in the container exceeds a threshold, and finally in the *trend* heuristic the container is burst when its potential is exhausted; during each access to the container the potential is incremented by a fixed amount when the root of the container is accessed, otherwise it is decremented by another amount. In our application we have chosen to follow a different set of heuristics since our main aim is to guarantee that IPs with common prefixes and similar distribution in their access characteristics will be grouped together in the same container under the same dominant IP; moreover the root of the container will correspond to the dominant IP of the group. Hence, storing IPs and employing these heuristics should guarantee that IPs that depict similar characteristics will be distributed in nearby containers, and thus it will be easy for our application to locate IPs that have the same prefixes and similar access characteristics, in order to efficiently locate the responsible sub network for a specific "burst" of visits.

More analytically our algorithm is as follows:

1. We store the IPs in the containers, with each container being implemented as a self-adjusting search tree.
2. We count as root accesses for a container the number of times the element of the root of the container is accessed. We also define as root ratio the one between the root accesses and the total number of accesses in a container.
3. The aim of using the Burst Trie is to cluster IPs with similar access behaviour under the same container (in order to capture their common prefix, and hence the latest sub network responsible for attacks). The ratio heuristic employed in (Heinz et al., 2002) is the one closest to our aims, but we found more appropriate to change it as follows:

   **Burst Rule:** Let $x$ be the number of elements in the container when it was initially created (after the split of another container). A container will be split again when the number of operations (accesses/insertions) becomes larger than $x$, and the root ratio is out of a chosen range $[l,u]$ that is a subrange of [0,1]. For example we could choose $[l,u]=[1/2, 2/3]$. Hence, when the number of operations in the container becomes greater than $x$ and the root ratio is smaller than 1/2 or larger than 2/3, it should burst.

   By employing the above burst rule, we try to guarantee that the elements in a container have similar access behaviour. Another technique to achieve that could be to maintain an access counter for each element of a container and split the container when the elements depict skewed access behaviour (the amount of how much skewed defined by us). However, this technique would cost too much and would need the employment of many parameters, and our burst heuristic seems more appropriate.

We take advantage of the previously described structure in order to detect suspicious networks for click fraud. Suppose that a "burst" of visits is generated from different IPs, which though belong to the same network. In the beginning, the first visiting IP may form its own container. As more and more visits from the same network occur, they are inserted in the same container and thus the Access trie (the upper level component of the burst trie) permits the efficient location of the common prefix of the IPs (the responsible sub-network), while the IP address in the container that caused the most visits will be splayed to the root of the container splay tree. After a certain number of visits coming from this particular network, the prefix of the IP denoting the network address will burst out of the containers due to the extreme amount of visits that cause the prefix's popularity to rise.

Using this technique, we are able to detect the responsible network for the "burst" of visits and subsequently take action to prevent a possible "burst" in the future coming from this particular network. Specifically, the moment that a "burst" of visits occurs, we detect the responsible IP and we then insert it in the burst trie. If the insertion of this IP in the burst trie results to an event of burst in the trie, then we identify the subnet corresponding to the extracted and this subnet is banned as suspect of click fraud. The banned network remains as is for a specific amount of time in order to avert any future attempts coming from this network. Only after this specific period of time has passed is the IP network allowed to access the Web pages again.

## 3.2 Example

Suppose that our Web page consists of five data units and that we store five timestamps for every data unit. Therefore, a 5x5 two dimensional matrix is formed (Table 1). A number of the visiting IPs is shown in Table 2.

Table 1: The two dimensional matrix that stores timestamps $t_{ij}$.

| | | | | |
|---|---|---|---|---|
| $t_{00}$ | $t_{01}$ | $t_{02}$ | $t_{03}$ | $t_{04}$ |
| $t_{10}$ | $t_{11}$ | $t_{12}$ | $t_{13}$ | $T_{14}$ |
| $t_{20}$ | $t_{21}$ | $t_{22}$ | $t_{23}$ | $T_{24}$ |
| $t_{30}$ | $t_{31}$ | $t_{32}$ | $t_{33}$ | $T_{34}$ |
| $t_{40}$ | $t_{41}$ | $t_{42}$ | $t_{43}$ | $t_{44}$ |

Suppose that all but the last of the example's IPs have already visited the Web site. Suppose now that at time $t_{04}$, the IP 150.140.141.2 visits the data unit 0. If by comparing the two timestamps $t_{00}$ and $t_{40}$, we realize that $t_{04}$- $t_{00}$ < T, then a burst has occurred. At the same time, the visiting IP is added in the burst trie. As shown in Figure 1, the addition of this particular IP results in the burst of a container and at the same time the total prefix that has burst out of the container has a length of 8 characters, including the ".". Thus, this prefix represents a subnet and the network 150.140 is banned.

Table 2: Table showing the visiting IPs.

| |
|---|
| 150.140.141.2 |
| 150.140.142.3 |
| 150.140.153.4 |
| 150.140.168.2 |
| 159.149.52.12 |
| 159.122.149.13 |

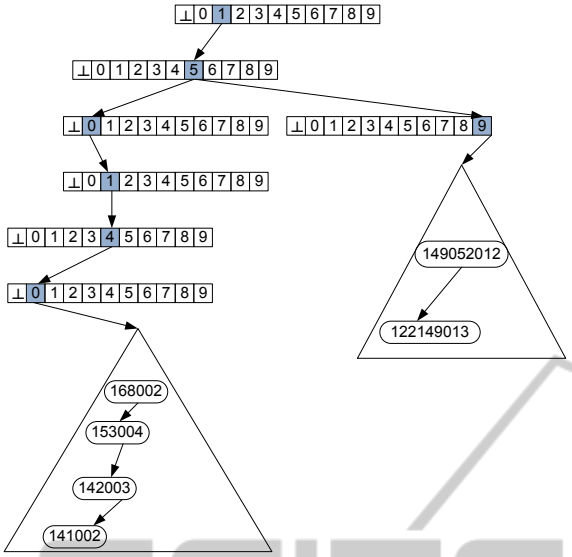


Figure 1: The burst trie after inserting the IP 150.140.168.2. (For readability reasons the example uses decimal representations instead of binary).

At this point, we ought to mention that, for readability purposes of the above example, we used the conjecture that a sub-network address (suffix) consists of six decimal digits, or equivalently 16 bits. The amount of bits that are used in order to declare the network address, which a certain IP address belongs to, may vary according to the subnetting method used. Thus, the conjecture adopted enables us to apply our algorithm in the general case, where 16 bits are used to represent the network address without loss of generality.

In order to test the efficiency of the algorithm in practice, the proposed technique was implemented and experimentally applied on the user traffic of an experimental test scenario. The technique was implemented, using C Sharp (C#) programming language, as a module (Burst Module) on Microsoft IIS (Internet Information Services) 7.5 Web Server, while utilizing the IHttpModule interface. Whenever an HTTP request arrives on the Web server, the relative IP address is forwarded and stored in the data structure proposed by our technique through the burst module, thus enabling the click fraud detection process.

In order to prove the effectiveness of the approach, Microsoft Visual Studio load test feature was utilized. Through the IP switching capability provided by Visual Studio (in each PC-test agent under the Load Test Manager an IP range(s) or random IPs are assigned thus enabling the emulation of a real life stress test scenario) we have conducted a range of experiments using a varying number of PCs (three, five, ten, fifteen) as test agents for the initial load test configuration and applying various real time scenarios:

- Mobile traffic, PC traffic etc.
- 100, 200, 500, 1.000, 3.000, 5.000, 10.000 requests per second.
- Various subnets assignments, as well as random IPs, to the various test agents' members etc.

The tests have proved in all cases that when a burst is detected, the Burst module automatically blocks the relative subnet causing the burst.

## 3.3 Calibration of the Algorithm - Parameters

As far as the size of the data structure is concerned, given that the algorithm is designed to be running in online mode, it should not exceed certain size constraints. One way to tackle this issue is to re-initialize the data structure. In other words, after a certain time period, the contents of the trie are cleared and a new version of the burst trie is built. The time period is chosen according to the visiting frequency of the Website on which the algorithm is running. In the case of high traffic Websites, the trie should be often re-initialized, unlike Websites with low traffic. Another way to cope with the increasing size of the burst trie would be a merge operation. Our future research includes studying how the nodes of the tree that their IP is not recently accessed could be merged, in order to avoid re-initializing the tree and decrease its size at the same time.

A second very important issue concerning the efficiency of the algorithm is the size of the sub-networks we wish to detect. We consider a subnet as a candidate to be banned when it is of class-B (16 bits prefix) and below in order to avoid banning a larger network for a click fraud coming from a subnet. To be able to detect the exact subnet from which the click fraud was generated, we have to find the common prefix of all IP-addresses (in binary) that visited the Web server.

For example, suppose that the Web server receives hits from the following IP-addresses:

- 150.140.141.8 (binary: 10010110. 10001100. 10001101. 00001000)
- 150.140.141.9 (binary: 10010110. 10001100. 10001101. 00001001)
- 150.140.141.10 (binary: 10010110. 10001100. 10001101. 00001010)
- 150.140.141.11 (binary: 10010110. 10001100. 10001101. 00001011

Then, a burst of visits is caused by the subnet 10010110. 10001100. 10001101.000010 (longest common prefix).

We apply our algorithm in order to detect the suspicious subnet. All IP-addresses are inserted into the burst trie. For achieving the best performance of the algorithm, it can be adjusted in several ways as follows:

1. If the IP-addresses are inserted in binary format, the alphabet of the trie would be {0, 1} and the number of the tree levels would increase. However, the exact subnet would be detected.
2. If the IP-addresses are inserted in decimal format, the alphabet of the trie would be {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} and although the tree would be more efficient, an accurate detection of the subnet would be impossible. The reason for this situation is that there is no one-to-one correspondence of the decimal representation to the binary one, as far as the subnets are concerned. If this is the case, the detection is limited to class A networks ($2^{24}$ IP-addresses), class B ($2^{16}$ IP-addresses) and class C ($2^{8}$ IP-addresses) resulting to non-efficient detection process and in most of the cases extreme IP-addresses ranges banning.

In order to find the best solution to the issues just described, a possible choice would be to insert IP-addresses in binary format but with all bits grouped in pairs. Then the trie alphabet would be {00, 01, 10, 11} and a better performance would be achieved, since the subnet detected would be close enough to the actual suspect subnetwork (maximum difference of one bit).

Another possibility would be to insert the IP-addresses in two formats simultaneously as follows. The fields of the address until class B network are inserted in decimal format (6 first digits: 255.255. ***. ***) and the remaining digits are inserted in simple binary format or as bits grouped in pairs. The reason for this choice is obvious, because as already explained the subnet we are looking for is of class B and below.

In all cases the splay trees of the burst trie data structure will host the remaining part of the IP addresses as a whole number (in binary or decimal format as shown on Figure 1).

## 4 CONCLUSIONS & FUTURE WORK

In this paper we have presented the design decisions for an algorithm for real-time detection of click fraud that focuses not on individual IPs, but on whole subnetworks. In order to achieve that, we exploited inherent properties and characteristics of the burst trie data structure that permit the efficient implicit clustering of IPs with common prefixes that depict similar access characteristics. It would be interesting to find out if this specific structure can have other applications too and compare it experimentally with other algorithmic choices that could (if suitably enhanced) depict similar characteristics, such as the structure in (Badr and Oommen, 2005).

Moreover, we are planning to investigate the use of other data structures that could be employed besides burst tries such as self-adjusting ternary search tries (Badr and Oommen, 2005), and compare the pros and cons of both alternatives.

## REFERENCES

Antoniou, D., Paschou, M., Sakkopoulos, E., Sourla, E., Tzimas, G., Tsakalidis, A. K. and Viennas, E.,2011.Exposing click-fraud using a burst detection algorithm. In *Proc. ISCC*, pages 1111-1116.

Badr, H. G., and Oommen, B., 2005. Self-Adjusting of Ternary Search Tries Using Conditional Rotations and Randomized Heuristics. In *Computer Journal*, pages 200-219.

Heinz, S., Zobel, J., and Williams, H. 2002. Burst tries: a fast, efficient data structure for string keys. In *ACM Trans. Inf. Syst*, pages 192-223.

Sakkopoulos, E., Antoniou, D., Adamopoulou, P., Tsirakis, N., Tsakalidis, A., 2010. A Web personalizing technique using adaptive data structures: The case of bursts in Webvisits. In *Journal of Systems and Software, Elsevier,* vol. 83, issue 11, pages 2200-2210.

Sleator, D. D, Tarjan, R.E., 1985. Self adjusting binary search trees. In *Journal of the ACM 32*, pages 652–686.

Zhou, B., Hui, S. C., Chang, K., 2004. An intelligent recommender system using sequential Web access patterns. In *IEEE Conference on Cybernetics and Intelligent Systems,* pages 393-398. IEEE, Singapore.