Wednesday, 25 November 2020

Bloom Filters and HyperLogLog in Network Security and IoT – A Review

Debaditya Sen
Department of Computer Engineering,
SRMIST, KTR, Chennai

Ritesh Bucha
Department of Computer Engineering,
SRMIST, KTR, Chennai

ABSTRACT

Undoubtedly, dealing with security issues is one of the most important and complex tasks various networks face today. A large number of security algorithms have been proposed to enhance security in various types of networks. Many of these solutions are either directly or indirectly based on *Bloom filter* (BF), a space- and time-efficient probabilistic data structure introduced by Burton Bloom in 1970.

IP networks are constantly targeted by new techniques of denial of service attacks (SYN flooding, port scan, UDP flooding, etc), causing service disruption and considerable financial damage. The on-line detection of DoS attacks in the current high-bit rate IP traffic is a big challenge. We propose in this paper an on-line algorithm for port scan detection. It is composed of two complementary parts: First, a probabilistic counting part, where the number of distinct destination ports is estimated by adapting a method called 'sliding HyperLogLog' to the context of port scan in IP traffic. The algorithm uses a very small total memory of less than 22 kb and has a very good accuracy on the estimation of the number of destination ports (a relative error of about 3.25%), which is in agreement with the theoretical bounds provided by the sliding HyperLogLog algorithm.

Keywords

Bloom filters, Security, Network processing, HyperLogLog, IP Traffic, SYN Flooding, Port Scanning

1. Bloom Filter in Network Security

a. INTRODUCTION

Security has always been a major concern for networked systems administrators and users. Many approaches have been proposed to achieve the various security goals. In these approaches, a variety of techniques and data structures have been used to address the security concerns in an efficient manner. On the other hand, there are typically umpteen numbers of data items that need to be stored, queried and updated in the network environment. Therefore, the fact is concluded that space and time are two important factors that should be taken into consideration by the security approaches, especially in the specific networks, such as sensor networks, which suffer from severe limitations. A probabilistic data structure that has been widely utilized in this field is Bloom filter (BF), which was introduced by Burton Bloom in 1970. BF is a simple, memory- and time-efficient randomized data structure for succinctly representing a set of elements and supporting set membership queries. These properties of BF make it very attractive to be utilized for many security applications. In recent years, BFs and their variants have been widely used in networking applications, such as resource routing, security, and web caching. This paper provides a survey on the applications of BFs in the field of network security. Note that we only focus on the idea behind the approaches without discussing implementation details. It should be noted, however, that our goal of making this survey is not providing an exact classification of security attacks for different networks. But we intend to review where BFs and their variants have been used to improve the efficiency of the different security schemes.

b. LITERATURE REVIEW

In this section, we review the network security schemes which are directly or indirectly based on BFs and their new variants. We conduct a taxonomy of uses of the BFs in different networks as shown in Table 1. In the wireless networks category, application of BFs in various types of wireless networks is discussed. In the other category, we study the BF applications in various fields related to wired networks, such as trace backing, pattern matching, and so on. The last three columns of this table give information about where Bloom filters have been embedded in each specific security application. These columns indicate whether the Bloom filters used for each security field are embedded in end-devices (ED), such as server machine, intermediate-devices (ID), such as routers or sensor nodes in the sensor networks, and/or in-packet (IP), where Bloom filter is located inside the packet traversing the network [20]. We emphasize that this categorization is not completely exhaustive. There may be some works that could be fallen into more than one category.

3.1. Wireless networks

In this section, the BFs applications related to various kinds of wireless networks are discussed.

3.1.1. Authentication Message authentication.

In [21], Son et al. proposed a communication-efficient message authentication protocol to authenticate messages flooded in large-scale WSNs. Each sensor node is preloaded with l symmetric keys and k hash functions. The sink also maintains k hash functions and n keys. The sink constructs n message authentication codes (MACs) using the n keys. These resulting MACs are then inserted into the BF. Subsequently, the BF is flooded along with the message in the whole network. When the message arrives at each node, l MACs are constructed again in the same way by using l keys stored in the node. These l MACs are sought in the arrived BF. When a zero value is found, the message is assumed to be invalid; otherwise, it is sent to the neighbour nodes [21]. Moreover, they proposed to use compressed Bloom filters [9] for reducing false positive rate and the size of BF. The scheme introduced in [22] tries to protect the data of the network from the attackers in an efficient manner. In this scheme, a group of sensors, named aggregators, classify the packets arrived from the other sensors. This scheme utilizes BFs for keeping trade-off between communication and computation costs and also enhancing the performance of the network. To this end, BF keeps the keywords associated with the nodes in

the network. The base station sends a request message in the form of Req [r, BFr (w1, ..., wn)] (where r is a random key and BFr (w1, ..., wn) is the BF resulting from k hash functions applied to the keyword r) either to the all nodes of the network or only to the aggregators. The sensors compare the keywords (their own BFs) with BFr (w1, ..., wn) to find a match. If there is a match, an encrypted message is generated based on the predefined policy and sent to the requester (aggregator or base station). One problem with this scheme is that the main consideration of it is to reduce the data redundancy; security is not taken into account sufficiently. In [23], Ren et al. proposed several public key cryptography based methods to provide a multi-user broadcast authentication service to minimize computation and communication costs. In the proposed method, called Bloom filter-based Authentication Scheme (BAS), all public keys assigned to the network users are inserted into a BF. Eventually, this BF is placed within all the sensor nodes of the network. When receiving a broadcast message, the sensor node checks the membership of its public key in the BF. Then, if a zero value is found, the message is discarded. The scheme is of interest but is applicable for special kind of WSNs with many user nodes.

Moreover, the whole scheme cannot resist DoS attack. In addition, the long time to verify each message using PKC increases the response time of the nodes.

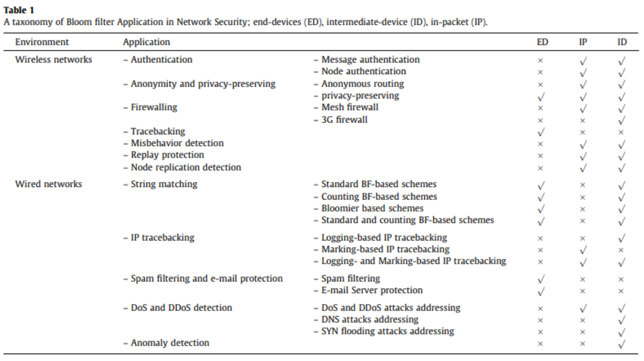

Table 1
A taxonomy of Bloom filter Application in Network Security; end-devices (ED), intermediate-device (ID), in-packet (IP).

Environment	Application		ED	IP	ID
Wireless networks	– Authentication	– Message authentication	×	✓	✓
		– Node authentication	×	✓	✓
	– Anonymity and privacy-preserving	– Anonymous routing	×	✓	✓
		– privacy-preserving	✓	✓	✓
	– Firewalling	– Mesh firewall	×	✓	✓
		– 3G firewall	×	×	✓
	– Tracebacking		✓	×	×
	– Misbehavior detection		×	✓	✓
	– Replay protection		×	✓	✓
	– Node replication detection		×	✓	✓
Wired networks	– String matching	– Standard BF-based schemes	✓	×	✓
		– Counting BF-based schemes	✓	×	✓
		– Bloomier based schemes	✓	×	✓
		– Standard and counting BF-based schemes	✓	×	✓
	– IP tracebacking	– Logging-based IP tracebacking	×	×	✓
		– Marking-based IP tracebacking	×	✓	×
		– Logging- and Marking-based IP tracebacking	×	✓	✓
	– Spam filtering and e-mail protection	– Spam filtering	✓	×	×
		– E-mail Server protection	✓	×	×
	– DoS and DDoS detection	– DoS and DDoS attacks addressing	×	✓	✓
		– DNS attacks addressing	×	×	✓
		– SYN flooding attacks addressing	×	×	✓
	– Anomaly detection		×	×	✓

c. METHODOLOGY

Information representation and query processing are two core problems of many computer applications, and are often associated with each other. Representation means organizing information according to some formats and mechanisms, and making information operable by the corresponding method. Query processing means making a decision about whether an element with a given attribute value belongs to a given set. For this purpose, BF can be an optimal candidate. A Bloom filter, conceived by Burton Howard Bloom in 1970, is a simple space-efficient randomized data structure for representing a set in order to support membership queries [1]. BFs may yield a small rate of false positives in membership queries; that is, an element might be incorrectly recognized as member of the set. Although Bloom filters allow false positives, for many applications the space savings and locating time constantly outweigh this drawback when the probability of false positive can be made sufficiently small. Initially, BF was applied to database applications, spell checkers and file operations [2–4]. In recent years, BFs have received a great deal of attention in networking applications, such as peer-to-peer applications, resource routing, security, and web caching [5,6]. A survey on the applications of Bloom filters in distributed systems can be found in [7]. BFs are also being used in practice. For instance, Google Chrome uses a Bloom filter to represent a blacklist of dangerous URLs. The idea of standard BF is to allocate vector A of m bits, initially all set to 0, for representing a set S = {x1, x2, ... , xn} of n elements. The BF uses k independent hash functions h1, h2, ... , hk, each with range {0, ... , m 1}. A BF is constructed in two phases: programming phase and querying phase [1,5]. In the programming phase, each element x 2 S is hashed by k independent hash functions. Then, all the bits at positions A[hi(x)] in A are set to 1 for (1 6 i 6 k). Fig. 1 depicts the pseudocode for insertion of n elements. A particular position in the vector might be set to 1 multiple times, but only the first time has an effect. In the querying phase, to query for an element y, we check the bits at positions hi(y). If any of the bits at these positions are 0, the element is definitely not in the set. Otherwise, either the element is in the set, or the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive. Fig. 2 depicts the pseudocode for querying an element. The more elements that are added to the set, the larger the probability of false positives. The percentage of false positive of a Bloom filter can be minimized by tuning the three parameters: (i) number of elements (n) added to generate the Bloom filter. In most cases, this parameter is defined by the application and, thus, cannot be controlled. (ii) Number of bits used in a Bloom filter (m). m can be used in order to minimize false positives but obviously the larger the value of m the less compact representation. (iii) Number of hash functions (k) used to create the Bloom filter. The larger k the higher processing overhead (CPU usage) especially if hash functions perform complex operations. Fig. 3 depicts the mentioned process. In Fig. 3, three elements x1, x2, and x3 are separately hashed by 3 hash functions and then the corresponding bits in A are set to 1. To check if the element y1 is in the set approximated by A, we check whether all A[hi(y1)] are 1. As depicted in Fig. 3, because the bit position 8 is not 1, we surely conclude that y1 is not a member of the set. Since all the three-bit positions related to y2 are set to 1, we conclude that y2 is a member, although this may be wrong due to the false positive probability. There is a trade-off between the probability of false positive and the length m of the BF array [1,5]. It has been proven that the probability of false positive (fp) is equal to:

$$fp = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1 - e^{-kn/m}\right)^{k} \tag{1}$$

```
Input: Set of $n$ elements;
Output: Bloom filter $A$;
$A$ = allocate $m$ bits initialized to 0;
for each $x_i$ in the set do
    for $j \leftarrow 1$ to $k$ do
        $A[h_j(x_i)] \leftarrow 1$;
```

Fig. 1. Pseudocode for programming phase.

```
Input: An element $y$;
Output: True/False;
for $j \leftarrow 1$ to $k$ do
    If $A[h_j(y)] = 0$ return False;
return True;
```

Fig. 2. Pseudocode for querying phase.

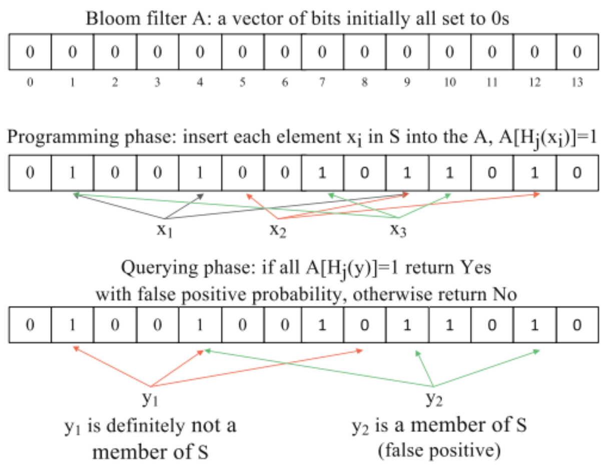
Bloom filter A: a vector of bits initially all set to 0s
Programming phase: insert each element x_i in S into the A, $A[H_j(x_i)]=1$
x_1
x_2
x_3
Querying phase: if all $A[H_j(y)]=1$ return Yes with false positive probability, otherwise return No
y_1
y_2
y_1 is definitely not a member of S
y_2 is a member of S (false positive)

Fig. 3. Insert and query operations in standard Bloom filter

d. SUMMARY ON LITERATURE SURVEY

In the last decade, Bloom filters have received a great attention in the network security area. This is because of their key features such as low memory requirement, high processing speed, low implementation complexity and the probabilistic nature of them. In this work, we provided an updated and comprehensive survey of the application of Bloom filters in various security application in both wired and wireless networks. Table 3 summarizes the contribution of various types of Bloom filters introduced in this paper to network security. For each variant, this table indicates its application domain and whether the false positives (FP) and/or false negatives (FN) are introduced (Yes/ No). We believe that Bloom filters will continue to be used in many new applications and also next variants of this structure will be introduced to deal with the incoming security problems.

e. REFERENCES

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

[2] M.K. James, Optimal semijoins for distributed database systems, IEEE Transactions on Software Engineering 16 (1990) 558–560.

[3] M.D. McIlroy, Development of a spelling list, IEEE Transactions on Communications 30 (1982) 91–99.

[4] L.L. Gremillion, Designing a bloom filter for differential file access, Communications of the ACM 25 (1982) 600–604.

[5] A. Broder, M. Mitzenmacher, Network applications of bloom filters: a survey, Internet Mathematics 1 (2003) 485–509.

[6] L. Fan, P. Cao, J. Almeida, A.Z. Broder, Summary cache: a scalable wide-area web cache sharing protocol, IEEE/ACM Transactions on Networking 8 (3) (2000) 281–293.

[7] S. Tarkoma, C.E. Rothenberg, E. Lagerspetz, Theory and practice of bloom filters for distributed systems, IEEE Communications Surveys and Tutorials 14 (1) (2012) 131–155.

[8] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, G. Varghese, An improved construction for counting bloom filters, in: Proceedings of the 14th Conference on Annual European Symposium, vol. 14, 2006, pp. 684–695.

[9] M. Mitzenmacher, Compressed Bloom filters, IEEE/ACM Transactions on Networking 10 (5) (2002) 604–612.

[10] B. Chazelle, J. Kilian, R. Rubinfeld, A. Tal, The Bloomier filter: an efficient data structure for static support lookup tables, in: Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 30–39.

2. Hyper loglog in Network Security

a. INTRODUCTION

Denial of service (DoS) attacks are one of the most important issues in network security. They aim to make a server resource unavailable by either damaging data or software or flooding the network with a huge amount of traffic. Thus, the server becomes unreachable by legitimate users, causing a significant financial loss in some cases. Port scan is a particular DoS attack that aims to discover available services on the targeted system. It essentially consists of sending an IP packet to each port and analysing the response to the connection attempts. Definitions found in the literature are enable to provide an absolute quantitative definition of port scan. The attack is rather defined by a comparison to the standard behavior. The attacker can discover not only available ports (or services) but also more relevant information about the victim such as its operating system, services owners, and the authentication method. Once the system vulnerabilities are identified, a future attack can be launched, engendering important damages. Various port scanning techniques have been developed and are very simple to install in order to launch serious port scan attacks. Nmap [2] is the most known port scan method. It was proposed by Fyodor in 2009. Zmap is a faster scanning method developed by Durumeric et al. in 2013. It can scan the IPv4 address space in less than 45 min using a single machine.

The memory size required by this kind of approach is proportional to the number of flows, which is clearly unscalable and not adapted to the current very high-bit traffic carried by very high-speed links. To overcome this problem, it is necessary to dispense accurate statistics and to generate estimates which require less memory and are based on a faster processing. In this context, some recent probabilistic methods based on Bloom filters have been proposed. A Bloom filter is an efficient data structure of a limited size that guarantees fast processing, thanks to the use of hash functions.

b. LITERATURE REVIEW

The sliding HyperLogLog algorithm is mainly based on the HyperLogLog algorithm [11] designed by Flajolet et al. in 2007. The HyperLogLog algorithm is a very efficient probabilistic algorithm for cardinality estimation. It performs a single pass on data and gives an accurate estimation of the number of distinct elements, using a very small memory. The HyperLogLog algorithm is nowadays used in many fields such as databases and networks, where an exact counting is impractical because of memory consumption and high latency. An example of an application of HyperLogLog for the improvement of database queries in Google systems is provided in [12]. A new release of the well-known database management system, PostgreSQL, has been recently developed in 2013 to add HyperLogLog data structures as a native data type (see [13] for more details). The HyperLogLog algorithm is said to be probabilistic because it uses some randomness introduced by the hash function h. The objective is to estimate the cardinality of a given set S, where elements can be repeated (also called multiset). Each element v of S will be first hashed into a random value h(v). The algorithm focuses on the following pattern ' 0ρ1' in the binary representation of h(v) (ρ is the position of the leftmost 1). Let us denote by R the highest value of ρ among all the elements of the multiset S. The key idea 0 of the algorithm is that with a larger cardinality, we are more likely to have a higher value of R. More precisely, 2R is a good estimator of the multiset cardinality. To have a more robust estimation, a stochastic averaging process is introduced. It consists in splitting S into m subsets S1,…S m , according the first b bits in the hashed value h(v), where m=2b. R[i] is computed and stored independently for each subset S i . R[1],…,R[m] are initialized to −∞ at the beginning.

R[i]=maxv∈Siρ(v),ρis the position of the leftmost 1.

$$Z := \left(\sum_{j=1}^{m} 2^{-R[j]} \right)^{-1}.$$

The harmonic mean of 2R[i],i∈{1,..,m}, is then computed using the following formula:

The normalized harmonic mean is given by

E:=(m∫∞0(log2(2+u1+u))mdu)−1.:=αmm2ZwhereαmE:=αmm2Zwhereαm:=m∫0∞log22+u1+umdu−1.

The full specification of the HyperLogLog algorithm is given by the following pseudo-code:

<u>The Hyper Loglog algorithm</u>

```
Assume $m = 2^b, b \in N$
Initialize the $m$ registers, $R[1], \ldots, R[m]$ to $-\infty$;

For $v \in S$ do
    - set $x := h(A)$;
    - set $i :=$ the subset identifier, given by the first $b$ bits of $x$;
    - set $x_{b+} := x$ stripped of its initial $b$ bits;
    - set $R[i] := max(R[i], \rho(x_{b+}))$, where $\rho(x_{b+})$ is the leftmost 1 position of $x_{b+}$;

Compute the harmonic mean of $2^{R[j]}$, $Z := \left(\sum_{j=1}^{m} 2^{-R[j]}\right)^{-1}$
return the estimate $E := \alpha_m m^2 Z$, where $\alpha_m := \left(m \int_0^\infty \left(\log_2\left(\frac{2+u}{1+u}\right)\right)^m du\right)^{-1}$.
```

The proposed method for port scan detection
The aim of this paper is to propose a new method that detects on-line port scan attacks in the IP traffic. Such a solution is designed to be implemented on a core router of the backbone network of the operator. The router has to analyze on the fly the huge amount of data received at a very high bit rate in order to extract relevant statistics that will be used by the decisional mechanism to identify the suspicious traffic. The two different steps of the algorithm are illustrated in Figure 1. The on-line analysis is a real challenge because the router has very limited resources and many other functions to provide.

c. METHODOLOGY

The aim of this paper is to propose a new method that detects on line port scan attacks in the IP traffic. Such a solution is designed to be implemented on a core router of the backbone network of the operator. The router has to analyze on the fly the huge amount of data received at a very high bit rate in order to extract relevant statistics that will be used by the decisional mechanism to identify the suspicious traffic. The two different steps of the algorithm are illustrated in Figure 1. The on-line analysis is a real challenge because the router has very limited resources and many other functions to provide.

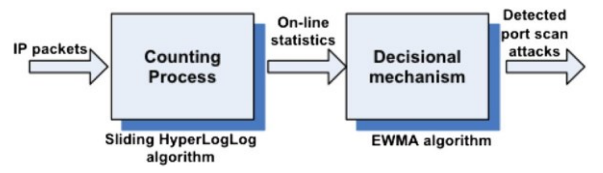

Figure 1 The proposed method for port scan detection.

Figure 1 The proposed method for port scan detection. Therefore, very efficient detecting methods are required to extract relevant statistics about the traffic and to make very quick decision about legitimacy of the traffic. In particular, the analysis of each packet has to be faster than the inter-arrival of the IP packets which is of only few nanoseconds.

Probabilistic counting method:

We focus in this paper on a particular kind of port scan attack called vertical port scan [8]. It consists of scanning many ports for a given destination. The number of destination ports can theoretically reach 65, 536 as it is encoded on 2 bytes, but the commonly used destination ports are not very numerous. The used destination ports are mainly composed of the so-called, in [1], wellknown ports (0-1023) and some registered ports (1024-49151). Therefore, the total number of distinct destination ports is a key observable to detect port scan attacks. In this context, the sliding HyperLogLog algorithm can be applied to count indefinitely, over a sliding window, the number of distinct destination ports. For each received packet (identified by the classical 5tuple composed of the source and destination addresses, the source and destination port numbers together with the protocol type), only the destination port will be considered and hashed into a random value. The sliding HyperLogLog algorithm will not perform an exact counting but will only provide an estimation. Therefore, a good choice of the parameters of the algorithm has to be done in order to ensure an acceptable error on the estimation. The number of buckets, m, is the crucial parameter of the counting method. With a high value of m, a smaller standard error can be achieved (1.04/ √m), but a larger memory will be used (Idsizemln(n/m) bytes). Moreover, m depends on the cardinality of the multiset: the number of distinct destination ports which can theoretically reach 65, 536. The number of distinct elements per bucket has to be high enough to perform significant statistics. With a total number of buckets of 1,024 (m = 1, 024 = 210), a standard error of only 3.25% can easily be achieved. Notice that this choice does not depend on the traffic trace and can be used for any port scan attack detection. There is clearly a tradeoff in the choice of the size of the sliding time window W. With a larger time window, one can answer requests concerning larger durations, for example, the number of destination ports in the last 30 min. But to deal with a larger time window, more information has to be stored. More precisely, the upper bound of the used memory (5mln(n/m) bytes) depends on n, the number of distinct destination ports, which is closely related to the size of the time window. Moreover, the standard duration of the attack must be considered in the choice of the size of the time window: W has to be large enough to notice the impact of the attack on the traffic.

Decisional mechanism:

Once relevant statistics related to port scan attacks are provided from the counting process, one has to filter and classify this information in two groups: 'standard behavior' and 'suspicious traffic'. This problem, also known as 'change-point detection' has been widely studied in the literature. Many methods have been developed by the community of statistics and data mining for several application fields.

Experimental results:

Dataset

The so-obtained algorithm with the two complementary parts (the counting and the decisional mechanisms) is tested against real IP traffic containing some port scan attacks. The considered traffic trace was captured in December 2007, by Orange Labs, in the context of an ANR-RNRT project on network security called 'OSCAR'. The traffic capture was performed on a router belonging to the IP backbone network of Orange Labs. Some global characteristics of this traffic trace are given in Table 1. The flow is defined as the set of those packets with the same source and destination addresses, the same source and destination port numbers, and the same protocol type.

Counting process

In this part, we focus on testing the counting process based on the sliding HyperLogLog algorithm. The number of distinct destination ports in the last time window W is estimated every 30 s. The time window is taken equal to 60 s, and the number of buckets m equals 1, 024. In Figure 2, the standard error on the estimated number of destination ports is plotted. One can notice that the standard error is most often within the theoretical

d. SUMMARY ON LITERATURE SURVEY

A new method identifying online port scan attacks in IP traffic is proposed. First, some relevant statistics are extracted from the data stream using the sliding HyperLogLog algorithm. A good choice of the parameters of this algorithm has to be done in order to ensure an acceptable accuracy of the statistics. Second, a change point detection method based on the EWMA algorithm is used to identify suspicious traffic. It is mainly an adaption of the EWMA to the data stream context. For this purpose, a learning phase is added at the beginning of the algorithm in order to initialize its parameters. Then, a new constraint is added to the moving average EWMA(t) update to overcome the false-positive problem when this latter exceeds the UCL detection threshold. Finally, we run experiments on a real traffic trace captured in the IP backbone network of Orange Labs in December 2007 in the context of the ANR-RNRT OSCAR project. The obtained results confirm the efficiency of the adapted combination of the Hyper

Loglog and EWMA algorithms in terms of accuracy, memory usage, and time response.

e. REFERENCES

[1] de Vivo M, Carrasco E, Isern G, de Vivo GO: A review of port scanning techniques. SIGCOMM Comput. Commun. Review 1999, 8: 411-430.

[2] Staniford S, Hoagland JA, Alerney McJM: Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. (Insecure, 370 Altair Way Ste 113 Sunnyvale, California 94086-6161 US).

[3] Durumeric Z, Wustrow E, Halderman JA: ZMap: Fast internet-wide scanning and its security applications. Paper presented at the 22nd USENIX security symposium. Washington, D.C., USA, 14–16 Aug 2013

[4] Jung J, Paxson V, Berger A, Balakrishnan H: Fast portscan detection using sequential hypothesis testing. Paper presented at IEEE symposium on security and privacy. Claremont Resort Oakland, California, USA, 9–12 May 2004.

[5] Mikians J, Barlet-Ros P, Sanjuas-Cuxart J, Sole-Pareta J: A practical approach to portscan detection in very high-speed links. Lect. Notes Comput. Sc 2011, 6579: 112-121. 10.1007/978-3-642-19260-9_12.

[6] Nam S, Kim H, Kim H: Detector SherLOCK: enhancing TRW with Bloom filters under memory and performance constraints. Computer Networks 2008, 52: 1545-1566. 10.1016/j.comnet.2008.01.008

[7] Sridharan A, Ye T, Bhattacharyya S: Connectionless port scan detection on the backbone. Paper presented at the 25th IEEE performance, computing, and communications conference (PCCC), Phoenix, AZ, USA, 0–12 April 2006.