EURASIP Journal on Information Security
a SpringerOpen Journal

RESEARCH Open Access

How can sliding HyperLogLog and EWMA detect port scan attacks in IP traffic?

Yousra Chabchoub*, Raja Chiky and Betul Dogan

Abstract

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. Second, a decisional mechanism is performed on the estimated number of destination ports in order to detect in real time any behavior that could be related to a malicious traffic. This latter part is mainly based on the exponentially weighted moving average algorithm (EWMA) that we adapted to the context of on-line analysis by adding a learning step (supposed without attacks) and improving its update mechanism. The obtained port scan detecting method is tested against real IP traffic containing some attacks. It detects all the port scan attacks within a very short time response (of about 30 s) and without any false positive. 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.

Introduction

Problem statement

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 [1]. It essentially consists of sending an IP packet to each port and analyzing 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 sevices) 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 [3] 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.

Network operators are always looking for scalable solutions to detect on-line DoS attacks. Their objective is to stop the attack very quickly in order to avoid wasting network resources. So, the attack detection solution should ideally be deployed very close to the source of the attack, which is unrealistic as it means that it has to be implemented for each user. Moreover, the DoS attacks can be launched from several sources against a single victim, at the same time, and are called distributed attacks (DDoS) in this case. To detect such attacks, one has to consider the aggregated traffic issued from the several sources, because the contribution of each source can be considered as a normal traffic. Therefore, the attack detection solution has to be implemented in a core router network so as to analyze the traffic issued from several users. It has also to perform an on-line analysis and rise alarms in case of

*Correspondence: yousra.chabchoub@isep.fr
ISEP, 21 rue d'assas, Paris 75006, France

suspicious traffic. In the context of core network, the real-time processing is a big challenge. In fact, the analysis time of an IP packet has to be shorter packet inter-arrival time which is of only some nanoseconds in the current IP traffic carried by core networks (8 ns in an OC-768 link). Moreover, as attack detection is not the main role of the router, which can provide many other functions such as prioritization and quality of service, the amount of memory used for attack detection has to be very small.

Related work

The problem of DoS attack detection in IP traffic has been largely addressed by the network security community. Most of the proposed methods analyze the exhaustive traffic and maintain accurate statistics about the various flows (number of packets per communication (source-destination pair), number of SYN packets sent by each source address, etc.) (e.g., the threshold random walk method proposed in [4]). 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 (see [5,6], and [7]). A Bloom filter is an efficient data structure of a limited size that guarantees fast processing, thanks to the use of hash functions.

Attack detection algorithms must first generate on-line aggregate information and statistics on the observed traffic, and then identify, based on the obtained statistics, the suspicious traffic that could correspond to attacks. Probabilistic algorithms can be used to extract statistics and estimates quickly, but they must be complemented by a decision phase that identifies attacks.

Various methods can be used for the decision phase. In [8], a detailed study of the port scan detection approaches is provided by Monowar et al. The different methods are divided into five main classes (soft computing, algorithmic, rule-based, threshold-based, and visual approaches). The performance of these algorithms is compared (accuracy, response time, etc.,). The results show that methods combining the data mining and threshold-based analysis are the most efficient in terms of false positive rates, robustness, and scalability. A common weakness of the threshold-based methods is that their accuracy is closely related to traffic characteristics. As an example, the detection mechanism used in [5], and [9], is based on the well-known problem of finding the top k elements from a data stream. This means that at most, k simultaneous attacks can be detected. Therefore, the parameter k must be well chosen in order to minimize false alarms and missed attacks. The aim of this paper is to use Bloom filters for the extraction of relevant information about the traffic and to automatically adapt a threshold-based algorithm to the on-line analysis context and the varying traffic conditions.

Organization of the paper

In this paper, we designed a new algorithm that detects on-line port scan attacks. The proposed method is mainly based on the sliding HyperLogLog algorithm [10] that we adapted to the context of port scan detection in IP traffic. Sliding HyperLogLog is an efficient algorithm that estimates the number of distinct elements over a sliding window. It is able to deal with a massive data stream and provides an accurate estimate using a very small memory. We used sliding HyperLogLog to analyze traffic and perform an on-line counting that we completed with a decisional mechanism that identifies port scan attacks.

The organization of this paper is as follows: The sliding HyperLogLog algorithm is presented in Section The sliding HyperLogLog algorithm. A detailed description of the proposed method for port scan detection is given in Section The proposed method for port scan detection. In this latter section, the counting method and the decisional mechanism are explained separately. The new detecting method is tested against experimental data collected from IP backbone network in Section Experimental results. It is also compared to other existing methods. Concluding remarks are presented in Section Conclusion.

The sliding HyperLogLog algorithm

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 '$0^\rho 1$' 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 of the algorithm is that with a larger cardinality, we are more likely to have a higher value of R. More precisely, 2^R 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 $S_1, \ldots S_m$, according the first b bits in the hashed value $h(v)$, where $m = 2^b$. $R[i]$ is computed and stored independently for each subset S_i. $R[1], \ldots, R[m]$ are initialized to $-\infty$ at the beginning.

$$R[i] = \max_{v \in S_i} \rho(v), \quad \rho \text{ is the position of the leftmost 1.}$$

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

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

The normalized harmonic mean is given by

$$E := \alpha_m m^2 Z \text{ where } \alpha_m := \left(m \int_0^\infty \left(\log_2 \left(\frac{2+u}{1+u} \right) \right)^m du \right)^{-1}.$$

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

The HyperLogLog algorithm

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 main advantage of the HyperLogLog algorithm is that it provides an excellent cardinality estimation, with a relative accuracy of about $1.04/\sqrt{m}$, using a very small memory equal to $m \log_2 \log_2(n/m)$ bits. m is the number of subsets, and n is the real cardinality of the multiset. In practice, using only 1.5 Kb, a cardinality of a one billion can be easily estimated with a typical standard error of about 2%.

The sliding window model is widely used in many applications requiring data stream management such as network monitoring, security, and financial applications. It consists of maintaining and updating some relevant statistics about the recent items of data stream. The sliding window can be logical or physical if it is, respectively, defined as the last N received items or the last time window T. Datar et al. [14] proposed a standard framework to adapt several applications (sums, averages, min, max, etc.) to the data stream context by adding a sliding window. They showed that their sliding window mechanism requires a memory overhead and adds a loss in the accuracy of the estimation. The additional error depends of course on the total used memory.

The sliding HyperLogLog algorithm proposed in [10] aims to adapt the original HyperLogLog algorithm to the context of data stream management. Its objective is to estimate the cardinality over a variable-bounded duration. In other words, one can answer at any time t the query about the number of distinct items seen over the last w units of time, where w is bounded by the window size W. For this purpose, it is compulsory to consider and store information about item arrival times (*timestamps*) to identify recent items at any time. Just like in HyperLogLog, each received item v will be hashed using the hash function h. Then, the corresponding subset will be identified with the first b bits of the hashed value. Finally, $\rho(v)$, defined as the position of the leftmost 1 in v stripped of its initial b bits, is computed. Thus, v will be associated to the pair $< t_v, \rho(v) >$, where t_v is the timestamp of the item v. The main idea of the sliding HyperLogLog algorithm is to maintain and update only relevant information, useful to answer at any time the query about the cardinality over the sliding window W. In particular, one must be able to compute, at any time, for each subset $i, i \in \{1, .., m\}$, the crucial parameter $R[i] = \max_{v \in S_i} \rho(v)$. For each subset i, a list called 'list of future possible maxima' (LFPM_i) will be stored. An element $< t_v, \rho(v) >$ remains in the LFPM_i if and only if it is a possible maximum over a future window of time. In other words, $R[i]$ can be equal to $\rho(v)$ for a future possible query at a future time t, concerning the last time window W. The LFPM_i is updated as follows:

For each received item $< t_v, \rho(v) >$, associated to the subset i, do the following:

- Delete old items (items with a timestamp $t < t_v - W$) from all the lists LFPM$_i$, $i \in \{1, .., m\}$;
- Delete items v' with $\rho(v') \leq \rho(v)$ from the LFPM$_i$;
- Add$< t_v, \rho(v) >$ to the LFPM$_i$.

To answer at a given time t, the query about the number of distinct elements seen over the last w units of time, one has to follow the following steps:
For each LFPM$_i$, $i \in \{1, .., m\}$

- Extract items concerned by the query: $< t_v, \rho(v) >$, with $t_v < t - w$.
- Compute $R[i]$ over the extracted items.

Compute the harmonic mean and the cardinality estimation with the exactly the same manner as that in the HyperLogLog algorithm.

Two major results about accuracy and memory consumption of sliding HyperLogLog algorithm are detailed in [10]. First, unlike in [14], adding the sliding window does not modify the accuracy of the algorithm. So, the accuracy of the sliding HyperLogLog algorithm is exactly the same as that in the HyperLogLog algorithm (a standard error of $1.04/\sqrt{m}$). Second, an upper bound to the total used memory was established. The total size of the m lists LFPM is bounded is $\mathrm{Id}_{\mathrm{size}} mln(n/m)$ bytes, where $\mathrm{Id}_{\mathrm{size}}$ is the size of the item identifier $< t_v, \rho(v) >$. In practice, the timestamp t_v is encoded on 4 bytes, and only 1 byte is sufficient for $\rho(v)$.

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.

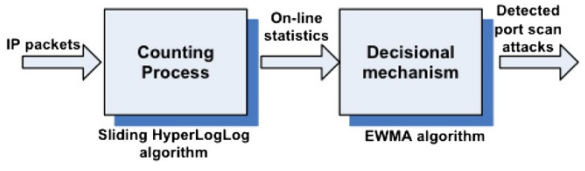

Figure 1 The proposed method for port scan detection.

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], well-known 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 ($1.04/\sqrt{m}$), but a larger memory will be used ($\mathrm{Id}_{\mathrm{size}} mln(n/m)$ 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 ($m = 1,024 = 2^{10}$), 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.

The port scan attacks last about 20 min and should be detected from the first minute. Thus, $W = 60$ s is a good choice for the size of the time window. So, the total used memory will be less than 22 kB, which is very reasonable for a router.

Some slow attacks, also called progressive attacks, are more difficult to detect because the intensity of the attack is increasing slowly. The attack lasts more than the standard duration in this case. To detect this kind of attack, one has to aggregate more the traffic in time. For the sliding HyperLogLog algorithm, a larger time window $W' = 5$ min can be added. The algorithm is performed independently and in parallel for the two time scales: $W = 60$ s and $W' = 5$ min.

Decisional mechanism

Once relevant statistics related to port scan attacks are provided from the counting process, one has to filter and classify these 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.

In their [15] book entitled *Detection of Abrupt Changes: Theory and Application*, Basseville and Nikiforov provided the description and the performance analysis of a wide range of algorithms dealing with this problem of change detection. They classified these algorithms into three main categories: the elementary algorithms, the cumulative sum algorithm, and the Bayes-type algorithms.

The elementary algorithms use simple and intuitive concepts. They have many industrial applications, in particular, in the quality control field. One can cite the Shewart control charts algorithm that was first introduced by Walter Shewhart [16] in 1924. This technique has found many applications in improving the quality of manufacturing processes [17]. A more efficient method, called 'the geometric moving average control charts' algorithm, was proposed later by Roberts in 1965 [18]. This algorithm is also known as the 'exponentially weighted moving average' (EWMA) algorithm. Its key idea is to give different weights to the values of the observed process, to detect the change point: the recent values must be given more importance. Another solution presented in the 'finite moving average' (FMA) is to ignore very old observations by using a finite set of weights. The filtered derivative algorithm is another elementary algorithm introduced by Basseville and Gasnier [19] in 1981, based on the gradient techniques. It is widely used in the context of image edge detection.

The 'cumulative sum' (CUSUM) algorithm was designed by Page in 1954 [20]. It is based on the sum of the process past observations. The CUSUM algorithm is well adapted to the detection of systematic small variations of the process.

The Bayesian algorithms were first introduced by Girshick and Rubin in 1952 [21]. The main advantage of these methods is that they guarantee a robust performance with a formal proof of optimality, but they need an *a priori* knowledge about the observed process, more precisely, the distribution of the change time must be given in advance.

Roberts presents in [22] a comparative experimental study of all the different algorithms described above. The input parameters of all these algorithms are set to their optimal values, and the mean detection delays are compared. The main result is that the CUSUM algorithm outperforms the other algorithms when the observed process has small shifts.

Recall that in the context of on-line port scan detection, we focus on change-point detection in a high-speed data stream. Sequential data is provided on the fly from the counting process, and our purpose is to identify as quickly as possible the change point using a very small memory. According to Basseville and Nikiforov [15], all the approaches described above can be used in the context of on-line analysis. But, in our particular context of port-scan detection, no assumption about the distribution of the attack time can be made. So, the Bayesian algorithms are not adapted for such applications. Moreover, the total duration of an attack is about 20 min, so the detection delay must be small enough (less than 1 min) to stop the attack quickly. Therefore, the lack of reactivity of the CUSUM algorithm against abrupt change points may be a big weakness.

Sebastiao and Gama performed in [23] an experimental comparison between some particular algorithms well adapted for an on-line analysis. More precisely, they compared the efficiency of the four following algorithms: the statistical process control (SPC) [24], the adaptative windowing (ADWIN) [25], the fixed cumulative windows model (FCWM) [26], and the Page-Hinkley test (PHT) [20]. These methods are closely related to the the three kinds of algorithms presented above. The main result of this paper is that PHT and SPC are less time- and memory-consuming, but in some cases, they endanger a high rate of false alarms.

To achieve our objectives in terms of detection delay, on-line analysis (only one pass over the whole data) and without any *a priori* knowledge about IP traffic characteristics, we choose to focus in this paper on an elementary algorithm: EWMA [18] proposed by Roberts. The main idea of the EWMA algorithm is to define a threshold delimiting a 'standard behavior' and to handle and update periodically an average of the observed data stream. The change point is then declared as soon as the average exceeds the fixed threshold. This algorithm is very simple

to implement. Unlike ADWIN, the past values of the observed data are not stored. It does not require any data structure. It also has a lower complexity because for each observed data, one has only to update a weighted average [27]. Compared to PHT and CUSUM, EWMA has the advantage to closely relate the importance of the observed data to its age which is more meaningful in the context of data stream. In PHT and CUSUM algorithms, all data history is equivalent and is considered in the same manner.

Our objective in this is to adapt the EWMA algorithm to the context of port scan detection and to experiment and evaluate the so obtained version. EWMA is an advanced control chart which calculates, over a sliding window, a weighted average of the data, taking into account the past observations. The older the observation, the less weight it has in the computation of the average. The sliding weighted average is updated as follows:

$$\begin{cases} \text{EWMA}(0) := \text{the average of all the observations} \\ \text{EWMA}(t) = \lambda * Y(t) + (1-\lambda) * \text{EWMA}(t-1), \text{ for } t > 0, \end{cases}$$

where $Y(t)$ is the observed value at time t. λ is a multiplicative factor ($0 < \lambda \leq 1$). It can be interpreted as a kind of correlation between $Y(t)$ and EWMA(t). In practice, we want the moving average EWMA(t) to follow carefully the variations of the observed process $Y(t)$. To give more importance to the past observations, than the current one, λ is very often taken smaller than 0.5. However, with very small value of λ, the algorithm becomes insensitive to some attacks having a small duration or a moderate intensity. That is why λ is usually between 0.2 and 0.5 in practice. EWMA(0), also called the target, is the average of the whole data set.

In this control chart, an alarm is raised as soon as the moving average EWMA(t) exceeds some thresholds called 'upper control limit' (UCL) and the 'lower control limit' (LCL), respectively, defined as

$$\text{UCL} = \text{EWMA}(0) + k * \sqrt{\lambda/(2-\lambda)s_0^2}$$

$$\text{LCL} = \text{EWMA}(0) - k * \sqrt{\lambda/(2-\lambda)s_0^2},$$

where the factor k is either set equal to 3 or chosen using the tables in Lucas et al. [28] in the enhanced version of EWMA proposed in 1990. s_0 is the standard deviation calculated on the whole data set.

It is clear that the so-described EWMA algorithm cannot be directly applied for the on-line data stream analysis because it clearly requires two passes over the whole data set. In fact, it needs first to compute EWMA(0) and s_0 using the whole data to fix the threshold UCL. Then, all the data will be considered again to detect the change points. To overcome this problem, we propose to add a learning step of some minutes at the beginning of the algorithm in order to initialize its parameters, namely, EWMA(0) and s_0. No change-point detection will be performed during this learning step. So, we implicitly assume that this period corresponds to the 'standard behavior' and does not contain any anomaly.

Notice also that in the EWMA algorithm described above, the moving average EWMA(t) is always updated even if it exceeds the detection threshold UCL. It turns out however that it is useless to update EWMA(t) in this latter case as this can generate some false positives (false alarms) just after some intensive attacks. In fact, when EWMA(t) gets very high, in case of an attack, it needs a long time (several slices of time windows) to reach its normal values (<UCL) when the attack is stopped. Thus, we propose the following update mechanism for EWMA(t):

$$\begin{cases} \text{if } (\text{EWMA}(t) > \text{UCL}) \text{ then } \text{EWMA}(t) = \text{EWMA}(t-1) \\ \text{else } \text{EWMA}(t) = \lambda * Y(t) + (1-\lambda) * \text{EWMA}(t-1). \end{cases}$$

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

Table 1 Characteristics of the traffic trace used for attack detection

Duration	No. of IP packets	No. of flows
67 min	32.10^6	250.10^3

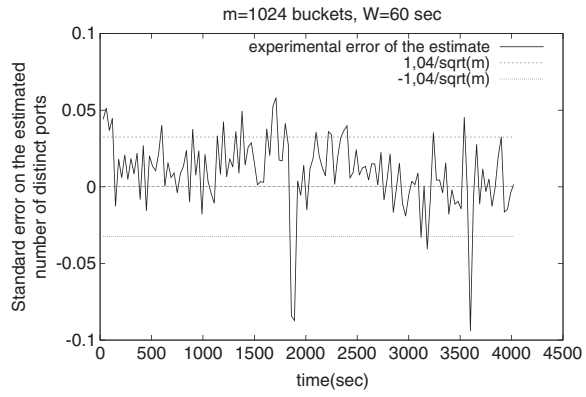

Figure 2 Standard error on the estimation of the number of distinct destination ports.

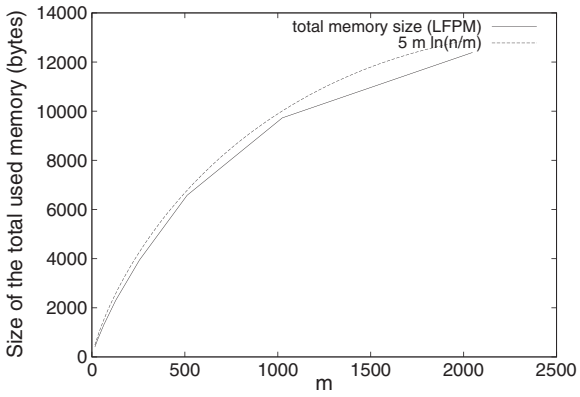

Figure 4 Impact of *m* on the total used memory (LFPM).

error σ of 3.25% ($\sigma = 1.04/\sqrt{m}$). It sometimes slightly exceeds this value, in fact σ is not an upper bound, but just an estimation of the standard error. As mentioned in [11], the estimate provided by HyperLogLog is expected to be within σ, 2σ, 3σ of the exact count in, respectively, 65%, 95%, and 99% of all the cases. Thus, the experiments confirm that adding the sliding window mechanism does not affect the accuracy of the original HyperLogLog algorithm.

In Figure 3, the size of the total used memory is plotted. It closely depends on n, the number of distinct destination ports. The results show that the total used memory (also called LFPM) is always below the theoretical bound of $5mln(n/m)$ bytes given in [10]. Recall that m is here constant (equal to $1,024$). This corresponds to a total used memory less than 22 kB.

Figure 4 shows the impact of m on the size of the total used memory. The mean size (calculated over the whole traffic trace) of the list LFPM is computed for different values of m, ($m = 16, 32, 64, \ldots 2,048$). One can easily check that the total used size is always below the theoretical bound. In the sliding HyperLogLog algorithm, for each received packet, the LFPM is updated in order to erase very old elements. So, the number of elements in the LFPM impacts directly the complexity of the update operation. One can conclude that the value of the parameter m can be chosen according to a total memory usage constraint. Moreover, with a good choice of m, it is possible to control the complexity of the algorithm.

In the description of the counting method, an additional time window ($W' = 5$ min) is suggested to deal with slow or progressive attacks. Recall that the time window was fixed to 1 min in the previous experiments. So, one has to check that the total used memory remains reasonable for larger time windows. For this purpose, many time windows have been considered in Figure 5 (W between 1 and 5 min). m is constant, taken as equal to $1,024$. In a larger time window, more distinct destination ports can be seen. So, more information will be stored in the list of future possible maxima (LFPM); that is why a larger memory is

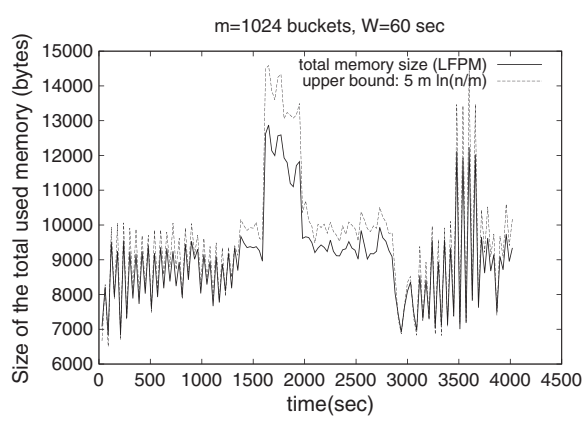

Figure 3 Size of the total used memory (LFPM).

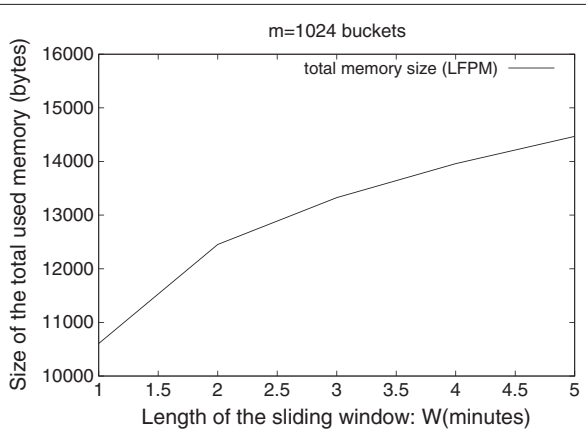

Figure 5 Impact of the sliding window *W* on the total used memory (LFPM).

required. But, the total used memory is still small as it is only multiplied by 1.5 when the time window grows from 1 to 5 min. This can be explained by the fact that the total number of distinct destination ports is bounded by $2^{16} = 65,536$. Moreover, many frequently used ports (well-known and registered ports) can be used over several 1-min time windows. This information is confirmed by Figure 6 which shows that the estimated number of distinct destination ports, with two time scales ($W = 1$ min and $W' = 5$ min).

Decisional mechanism using EWMA algorithm

The input of this decisional part is the estimated number of distinct destination ports provided by the counting process. This information is received every 30 s; it concerns the last 60-s time window. The input data is presented in Figure 6. One can easily see several peaks that are very likely to correspond to some port scan attacks. Our objective here is to automatically identify these peaks using the EWMA algorithm. The multiplicative update factor λ is taken equal to 0.3 and the detection parameter k equals to 3. All the implementations are performed with R.

In Figure 7, the original version of EWMA algorithm is tested. We considered first the whole data to compute the parameters of the algorithm: the target EWMA(0), the standard deviation S_0, and thus the upper bound UCL. A second pass on the data is performed to detect the port scan attacks. EWMA(t) is plotted; it is updated every 30 s. The detected port scan attacks are represented by squares: An alarm is raised each time EWMA(t) exceeds the upper bound UCL. We are not interested here in the lower bound LCL defined by the EWMA algorithm. Notice finally that the attacks are detected within a reasonable delay time of only 30 s.

Figure 8 concerns the first improvement added to the EWMA algorithm: To respect the on-line analysis constraint, only one pass on the data is performed. First

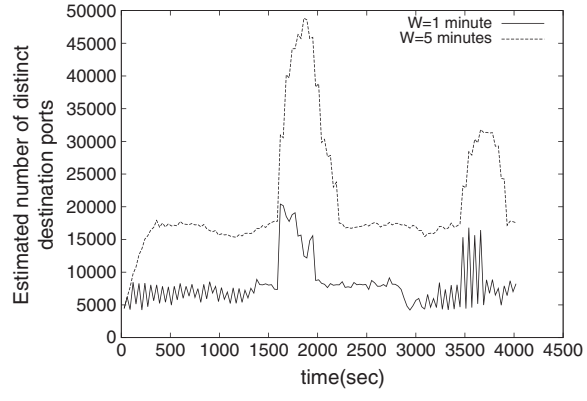

Figure 6 Input data: the estimated number of ports over 1- and 5-min sliding windows.

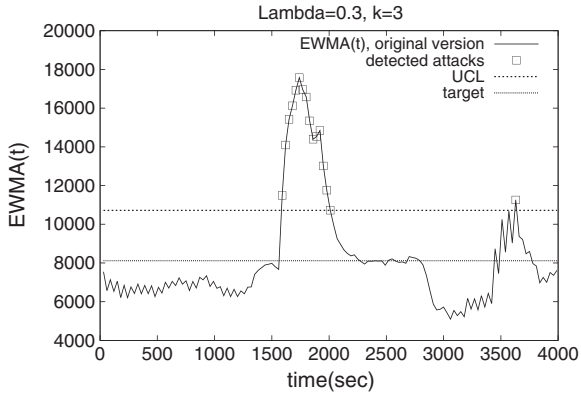

Figure 7 The original version of the EWMA control chart.

EWMA(0), S_0, and UCL are calculated in the introduced learning step taken equal to 10 min. Notice that the target EWMA(0) is lower than in the original version of EWMA. In fact, unlike in the original version, the attacks (where the observed data has high values) are not included in the computation of $EWMA(0)$. So, with this proposed improvement, more attacks can be detected because UCL has also a lower value as it is closely related to EWMA(0).

In Figure 9, the second improvement is added: The weighted moving average EWMA(t) is not updated in case of attacks. It is constant, keeping its old value. The squares correspond to the calculated values of the weighted moving average, and as they exceed UCL, they are not affected by EWMA(t). So, in case of attack, EWMA(t) and thus the squares (calculated using EWMA(t)) have lower values than in the original version of the EWMA. This enables EWMA(t) to reach very quickly the normal behavior (<UCL) when the attack stops. That is why we have less alarms in this case (compared to Figure 8), in particular at the end of the attacks.

The same experiments as that in Figure 9 are performed with different values of λ in order to analyze

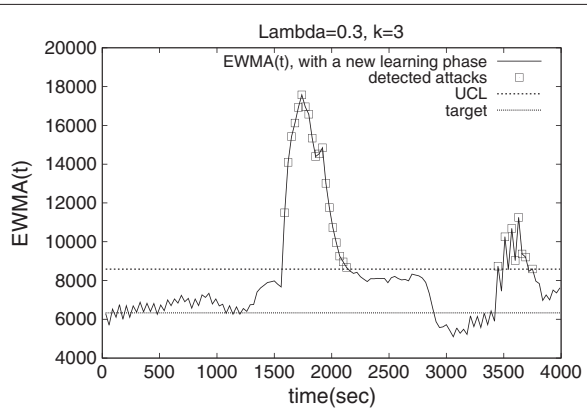

Figure 8 The EWMA control chart with a new learning step.

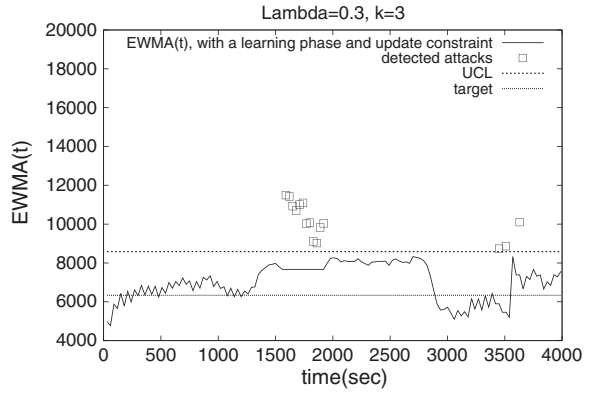

Figure 9 The EWMA control chart with a new learning step and an additional update constraint, $\lambda = 0.3$.

the impact of this parameter on the performance of the algorithm. Recall that λ is used to update the moving average EWMA(t) and to compute the upper bound UCL $=$ EWMA$(0) + k*\sqrt{\lambda/(2-\lambda)s_0^2}$. In Figure 9, λ is taken equal to 0.3. With a value of λ set to 0.5, roughly, the same results are obtained: the target EWMA(0) is always equal to $6,333$ as it does not depend on λ. The upper bound, UCL, is higher ($9,297$ instead of $8,458$), but the same alarms are raised at the same moments. Figure 10 shows the obtained results for $\lambda = 0.1$. One can notice that in this case, UCL is very close to the target EWMA(0); it is equal to $7,441$. In fact the difference UCL$-$EWMA$(0) = k*\sqrt{\lambda/(2-\lambda)s_0^2}$ is proportional to $\sqrt{\lambda/(2-\lambda)}$, which is an increasing function on $[0:1]$ with values in $[0:1]$. In this case, only small variations of EWMA(t) are tolerated, and many false positives (false alarms) are generated by the algorithm. Thus, the results are consistent with the choice of λ (between 0.2 and 0.5) presented in the description of the EWMA algorithm.

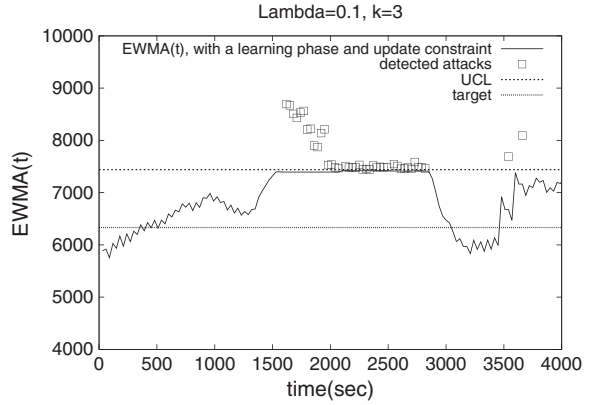

Figure 10 The EWMA control chart with a new learning step and an additional update constraint, $\lambda = 0.1$.

The use of the learning phase aims to adapt on-line the parameters of the algorithm to the initial characteristics of the traffic. EWMA(0) and UCL only depend on the traffic of this training set. It is clearly assumed that this period (taken equal to 10 min in practice) corresponds to a standard data having no attacks. The performance of the algorithm is of course closely related to the effectiveness of this assumption. We propose the following solution to reduce the dependance on this assumption and improve the robustness of the algorithm: We define a lower bound LCL in the same manner as in the EWMA algorithm:

$$\text{LCL} = \text{EWMA}(0) - k*\sqrt{\lambda/(2-\lambda)s_0^2}$$

If, at any time t, EWMA$(t) <$ LCL, then we restart the algorithm. The idea is that if we consider the worst case where the learning phase contains many port scan attacks, EWMA(t) will have smaller values at the end of the attacks, which can be detected by comparing EWMA(t) to the lower bound LCL defined in the learning phase. In other words, if there is no more correlation between the initial parameters calculated on the training set and the current values, the algorithm has to be restarted again. The learning phase is very useful as there is no absolute quantitative description of a port scan attack. The attack is simply defined as a significant deviation from a standard behavior that has to be learned online.

EWMA versus CUSUM algorithm

The CUSUM algorithm was first introduced by Page in 1954 [20]. It aims to detect small process shifts and is based on the cumulative sum C_t of the observed process $Y(i)$:

$$\text{CUSUM}(t) = \sum_{i=1}^{t}(Y(i) - \mu_0) = \text{CUSUM}(t-1) + (Y(t) - \mu_0),$$

where μ_0 is the target of the process given by its mean value. A process shift is declared if CUSUM(t) exceeds the upper bound $k*\sigma$, σ being the standard deviation of the process $Y(i)$. Notice that in this original version of the CUSUM algorithm, σ and μ_0 are calculated on the whole process. Therefore, in our case, two passes on the dataset are required to detect the port scan attacks using the original version of CUSUM. This version is clearly not adapted to an online analysis. Figure 11 shows the result of the CUSUM algorithm on the dataset containing some port scan attacks. The CUSUM algorithm was implemented with R. According to the recommendations given in [27], the process $Y(i)$ was first normalized, and the CUSUM algorithm was applied on the $X(i)$ process defined by the following relation:

$$X(i) = \frac{Y(i) - \mu_0}{\sigma},$$

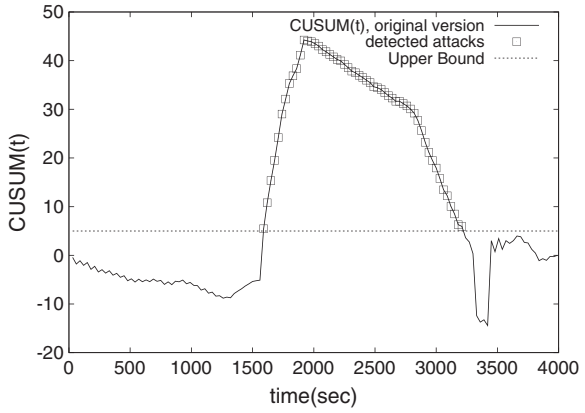

Figure 11 The original version of CUSUM control chart.

$X(i)$ has a mean of 0 and a standard deviation equal to 1. The upper bound k is taken equal to 5 to have good ARL properties as mentioned in [27]. Figure 11 shows that only the first attack is detected. The second attack that happens around the second 3,450 has a shorter duration and a smaller magnitude; that is why it has a limited impact on CUSUM(t) and can not be detected. Moreover, the duration of the first attack is largely overestimated. In fact CUSUM(t) is updated even in case of attacks, and unlike EWMA(t), in its original version, it takes a long time to reach normal values at the end of the attack. This can be explained by the fact that the CUSUM algorithm accumulates the effect of the attack over several sliding windows as it is based on a sum and gives the same weight to all the observed values.

Just like the EWMA algorithm, we introduced a learning step to adapt the CUSUM algorithm to the online analysis context. Thus, μ_0 and σ computation is only based on the first 10 min. Moreover, CUSUM(t) is not updated in case of attack. The results of the so-obtained version of CUSUM are given by Figure 12. The attack that

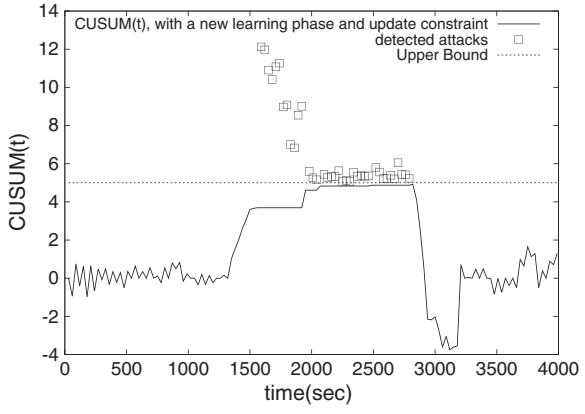

Figure 12 The CUSUM control chart with a new learning step and an additional update constraint.

happens at the end of the traffic trace (around the second 3,450) is not detected. In addition, between the second 2,000 and 3,000, the mean of the input process is slightly higher, which is considered by the CUSUM algorithm as an attack. One can easily check that it is a false positive using Figure 6 which displays the aggregated input data over larger time window. In fact, the time aggregation has no additive impact on the observed data. As a conclusion, the EWMA algorithm is more adapted than the CUSUM algorithm to the context of port scan attack detection. It is more reactive to the attacks with a short duration and a relatively small magnitude, and it is less sensitive to the small shifts that are mainly related to the varying IP traffic conditions (activity depending on the time of the day, etc.)

Conclusion

In this paper, 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 the 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 HyperLogLog and EWMA algorithms in terms of accuracy, memory usage, and time response.

Competing interests
The authors declare that they have no competing interests.

Received: 24 July 2013 Accepted: 21 January 2014
Published: 7 February 2014

References

1. M de Vivo, E Carrasco, G Isern, GO de Vivo, A review of port scanning techniques. SIGCOMM Comput. Commun. Review. **8**, 411–430 (1999)
2. S Staniford, JA Hoagland, 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. Z Durumeric, E Wustrow, 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. J Jung, V Paxson, A Berger, 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. J Mikians, P Barlet-Ros, J Sanjuas-Cuxart, J Sole-Pareta, A practical approach to portscan detection in very high-speed links. Lect. Notes Comput. Sc. **6579**, 112–121 (2011)
6. S Nam, H Kim, H Kim, Detector SherLOCK: enhancing TRW with Bloom filters under memory and performance constraints. Computer Networks. **52**, 1545–1566 (2008)
7. A Sridharan, T Ye, S Bhattacharyya, 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
8. H Monowar, DK Bhattacharyya, JK Kalita, Surveying port scans and their detection methodologies. Comput. J. **54**(10), 1565–1581 (2011)
9. C Levy-Leduc, Detection of network anomalies using rank tests. Paper presented at the EUSIPCO, Laussane, Switzerland, 25–29 Aug 2008
10. Y Chabchoub, G Hebrail, Sliding HyperLogLog: estimating cardinality in a data stream over a sliding window. Paper presented at the ICDM workshop on large-scale analytics for complex instrumented systems (LACIS), Sydney, 13 Dec 2010
11. P Flajolet, E Fusy, O Gandouet, F Meunier, Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. Paper presented at the 13th conference on analysis of algorithm (AofA), Juan des Pins, 17–22 June 2007, 127–146
12. S Heule, M Nunkesser, A Hall, HyperLogLog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. Paper presented at the EDBT 2013 conference, Genoa, Italy, 18–22 March 2013
13. GitHub, PostgreSQL extension adding HyperLogLog data structures as a native data type. http://github.com/aggregateknowledge/postgresql-hll. Accessed 20 Dec 2013
14. M Datar, A Gionis, P Indyk, R Motwan, Maintaining stream statistics over sliding windows. SIAM J. Comput. **31**(6), 1794–1813 (2002)
15. M Basseville, I Nikiforov, *Detection of Abrupt Changes: Theory and Application*, (Prentice-Hall, Upper Saddle River, 1993)
16. W Shewhart, *Bell Telephone Laboratories series: Economic Control of Quality of Manufactured Product*. (D. Van Nostrand Company, Princeton, 1931)
17. GEP Box, G Jenkins, *Time Series Analysis, Forecasting and Control*. (Holden-Day, San Francisco, 1990)
18. SW Roberts, Control chart tests based on geometric moving averages. Technomearics. **1**, 239–250 (1959)
19. BE M Basseville, J Gasnier, Edge detection using sequential methods for change in level. Trans. Acoust. Speech Signal Processing. **29**, 24–31 (1981)
20. ES Page, Continuous inspection schemes. Biometrika. **41**, 100–115 (1954)
21. M Girshick, H Rubin, A Bayes approach to a quality control model. Ann. Math. Stat. **23**, 114–125 (1952)
22. S Roberts, A comparison of some control chart procedures. Technometrics. **8**(3), 411–430 (1966)
23. R Sebastiao, J Gama, A study on change detection methods. Paper presented at the 14th Portuguese conference on artificial intelligence (EPIA), Aveiro, Portugal, 12–15 Oct 2009
24. J Gama, P Medas, G Castillo, P Rodrigues, Learning with drift detection, in *Advances in Artificial Intelligence*, ed. by ALC Bazzan, S Labidi (Springer New York, 2004), pp. 286–295
25. A Bifet, R Gavalda, Learning from time-changing data with adaptive windowing. Paper presented at the 7th SIAM international conference on data mining, Minneapolis, MN, USA, 29 Sept–1 Oct 2007
26. R Sebastiao, J Gama, Monitoring incremental histogram distribution for change detection in data Streams, in *Lecture Notes in Computer Science: Knowledge Discovery from Sensor Data*, ed. by MM Gaber, RR Vatsavai, OA Omitaomu, J Gama, NV Chawla, and AR Ganguly (Springer New York, 2010), pp. 25–42
27. C Douglas, *Introduction to Statistical Quality Control*. (Wiley, New York, 2004)
28. JM Lucas, MS Saccucci, RVJ Baxley, WH Woodall, HD Maragh, FW Faltin, GJ Hahn, WT Tucker, JS Hunter, JF MacGregor, TJ Harris, Exponentially weighted moving average control schemes: properties and enhancements. Technometrics. **32**, 1–29 (1990). doi:10.1080/00401706.1990.10484583

doi:10.1186/1687-417X-2014-5
Cite this article as: Chabchoub *et al.*: How can sliding HyperLogLog and EWMA detect port scan attacks in IP traffic? *EURASIP Journal on Information Security* 2014 **2014**:5.