Adaptive algorithms for identifying large flows in IP traffic

Yousra Chabchoub, Christine Fricker, Fabrice Guillemin, Philippe Robert

Adaptive algorithms for identifying large flows in IP traffic

Yousra Chabchoub, Christine Fricker, Fabrice Guillemin, and Philippe Robert

***Abstract*— We develop in this paper an adaptive algorithm based on Bloom filters in order to identify large flows. While most algorithms proposed so far in the technical literature rely on a periodic erasure of the Bloom filter, we propose in this paper to progressively decrement the various counters of the filter according to some overload criteria. When tested against real traffic traces, the proposed algorithm performs well in the sense that a high percentage of large flows in traffic are detected by the algorithm. In order to improve the accuracy of the algorithm, we introduce a shadow Bloom filter, which is less frequently decremented so that elephants have more chance of being identified. Since elephant detection issue is very close to flood attack detection, we adapt the proposed algorithm in order to detect SYN and volume flood attack in Internet traffic. The attack detection algorithm is tested against traffic traces from France Telecom collect and transit networks. Some performance issues are finally discussed.**

***Index Terms*— Bloom filter, TCP, flows, attack detection**

I. Introduction

In the characterization of Internet traffic, it is commonly admitted to distinguish two kinds of flows: small flows (i.e. having less than C packets, say), referred to as mice, and longer flows called elephants. These latter carry the main part of traffic (in Bytes) but most of flows are mice. It is often to observe that 95 % of flows are mice but elephants offer more than 95 % of global volume. Mice are typically generated by web browsing while elephants are due to file transfers (ftp, peer-to-peer , etc.). For traffic engineering purposes (e.g., supervision, security, etc.), it is important to design *on-line* algorithms to identify elephants and estimate their statistics. In the case of security (e.g, Denial of Service detection), flows are characterized in terms of the number of packets instead of their volume. Because such algorithms are typically implemented in network elements (routers or Deep Packet Inspection devices) running at very high speed, limited computing capacity and memory (several Mega-Bytes) are available to implement them. See Papagiannaki *et al.* [1] and Estan and Varghese [2] for example.

Due to the very high bit rate and the huge number of flows, it is unrealistic to expect an exhaustive description of the set of active flows. Indeed, maintaining the list of active flows and updating counters for each of them is hardly possible in an on-line context. Consequently, only an estimation of the characteristics of elephants can be expected within these constraints. In this paper we study the problem of identifying on the fly long flows of an intense IP traffic.

An important motivation of this paper is to study in depth the impact of the variations of traffic on on-line algorithms designed to estimate and control IP traffic. Detecting long flows is the main problem considered in this paper, but it can also be seen as a (simple) example exhibiting the importance for on-line algorithms to adapt to traffic variations. Quite often in the technical literature, algorithms depend (sometimes in a hidden way) on constants directly related to the traffic intensity. A same algorithm, which can run very satisfactorily in some instances, can also fail if the constants are unadapted to traffic characteristics. It is in fact highly desirable that the various values used by algorithms automatically adapt, as simply as possible, to the varying traffic conditions. In our view, this is a crucial issue which is, sometimes, underestimated. In a quite different domain, a very classical and successful example of such a situation is the way the TCP protocol adapts the throughput of connections based on the congestion of the network (through the number of packet losses).

A natural solution to cope with the huge amount of data generated by the IP traffic is to use hash tables. A data structure using hash tables, a *Bloom filter*, proposed by B. Bloom [3] in 1970, has been used to test whether an element is a member of a given set. Bloom filters have been used in various domains: database queries, peer-to-peer networks, packets routing, etc. See Broder and Mitzenmacher [4] for a survey. Bloom filters have been used by Estan and Varghese [5] and Azzana [6] to detect long flows, see the discussion below.

A Bloom filter consists of k tables of counters indexed by k hash functions. The general principle is the following: for each table, the flow ID of a given packet (that is the addresses and port numbers of the source and the destination) is hashed onto some entry and the corresponding counter is incremented by 1. Ideally, as soon as a counter exceeds the value C, it should be concluded that the corresponding flow has more than C packets. Unfortunately, since there is a huge number of small flows, it is very likely for instance that a significant fraction (i.e. more than C for example) of them will have the same entry, incrementing the same counter, thereby creating a false long flow.

To avoid this problem, Estan and Varghese [5] proposes to reinitialize counters to zero periodically. But without any a priori knowledge on traffic (intensity, flow arriving rate, etc.), the refreshment frequency can either be very low (in which case the filters can be saturated, and many false positives can be caused by mice) or very high (a significant fraction of elephants can be missed in this case). Therefore, the accuracy

Y. Chabchoub, C. Fricker and Ph. Robert are with INRIA Paris — Rocquencourt, Domaine de Voluceau, 78153 Le Chesnay, France. Email: First-Name.Last-Name@inria.fr

F. Guillemin is with Orange Labs, 2 Avenue Pierre Marzin, 22300 Lannion, France, Email: Fabrice.Guillemin@orange-ftgroup.com

of the algorithm closely depends on the period T of the refreshment mechanism of the filters, which is clearly related to the traffic intensity.

In this paper an algorithm based on Bloom filters with an additional structure, the virtual filter, and an adaptive refreshment mechanism is proposed. See Azzana [6]. As shown in the following, the algorithm notably improves the accuracy of algorithms based on Bloom filters. Moreover, related ideas can be applied in other domains like attack detection.

An interesting application field of these methods is the detection of attacks by denial of service. During such an attack, a machine is the target of a huge number of small flows coming from numerous machines connected to the network. An on-line identification of such anomalous behavior is necessary for a network administrator to be able to react quickly and to limit the impact of the attack on the victim. Via an adequate aggregation, the problem is boiled down to the detection of very long flows. Here again, adaptive properties of the detection algorithms to traffic conditions are essential to distinguish between normal variations of traffic and attacks. A related algorithm using Bloom filters with an adaptive refreshment mechanism is also proposed in this case.

The organization of this paper is as follows: A detailed description of the algorithm is given in Section II. The algorithm proposed is tested against experimental data collected from different types of IP networks in Section III. The application to the detection of denial of service (DoS) attacks is developed in Section IV. Some Performance issues of the algorithm is discussed in Section V. Some concluding remarks are presented in Section VI.

II. Algorithms based on Bloom Filters

In this section, we describe the on-line algorithm used to identify long flows and estimate their volume. We recall the usual definition of a flow: It is the set of those packets with the same source and destination IP addresses together with the same source and destination port numbers (and of course the same protocol type). In the following, we shall consider TCP traffic only.

To simplify the notation, long flows will be sometimes referred to as elephants and small flows as mice. For several reasons, this dichotomy is largely used in the literature, see the discussion in Papagiannaki *et al.* [1] for example.

Definition 1 (Mouse/Elephant): A mouse is a flow with less than C packets. An elephant is a flow with at least C packets.

The constant C is left as a degree of freedom in the analysis. Depending on the target application, C can be chosen to be equal to a few tens up to several hundreds of packets. The choice of C is left to the discretion of the operator.

Note that to estimate the *total* number of flows in efficient, nearly optimal, way, several on-line algorithms have recently been developed. See Flajolet *et al.* [7] and Giroire and Fusy [8]). Unfortunately, these algorithms do not seem to apply to identify the flows with more than C packets for some fixed C.

A. Description

The algorithm is based on the Bloom filter designed by Estan and Varghese [5]. The filter, see Figure 1, consists of k stages. Each stage $i \leq k$ contains m counters taking values from 0 to C. It is assumed that k independent hash functions h_1, h_2, ..., h_k are available. The total size of the memory used for the filter is denoted by M, recall that M should be of the order of several Mega-Bytes. An additional auxiliary memory is used to store the identifiers of detected elephants.

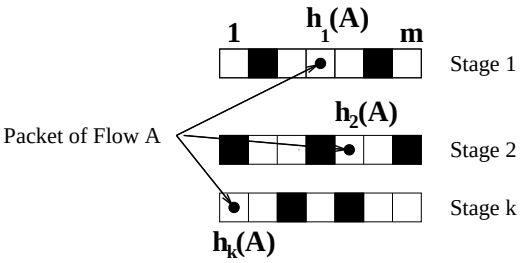

Fig. 1. The Bloom filter

The algorithm works as follows: All counters are initially set to 0; if a packet belonging to a flow A is received then:
— If A is in the memory storing elephant IDs.
Next packet.
— If not, let $\min(A)$ the minimum of the values of the counters at the entries $h_1(A)$, ..., $h_k(A)$ of the k hash tables.

- — If $\min(A) < C$, all the corresponding counters having the value $\min(A)$ are incremented by 1.
- — If $\min(A) = C$, the flow of A is added to the memory storing elephant IDs. The flow is detected as to be an elephant.

The algorithm as such is of course not complete since small flows can be mapped repeatedly to the same entries and create false elephants. One has therefore to clear the filters from the influence of these undesirable flows. Estan and Varghese [5] proposed to erase all counters of the filters on a periodic basis (5 seconds in their paper):

Estan and Varghese's refreshing mechanism

- — Every T time units:
  All counters are re-initialized to 0.

Ideally, the constant T should be directly related to the traffic intensity, otherwise if the refreshment mechanism occurs frequently, the counters corresponding to real elephants are decreased too often and a significant fraction of them may not reach the value C and therefore many elephants will not be detected. On the other hand, if T is too large then, because of their huge number, small flows may be mapped onto the same entry and would increase the corresponding counter to the value C, creating a false elephant. Estan and Varghese's algorithm could perfectly work if there would be a way to change the value of T according to the order of magnitude of the number of small flows. Such a scheme is however not clear to us. See Section III for experiments.

Our approach to refresh hash tables is based on the current state of the filters and not bound to some fixed time scale.

Adaptive Refreshing Mechanism

— If the state of filters is declared as overloaded then all positive counters are decremented by one.

Note that one only decreases by 1 the values of the counters instead of setting them all to 0. The idea is that if the overload condition of filters is properly chosen, then most of the values of the non-zero counters will be 1 or 2 when it holds. Remember that, structurally, compared to the number of mice, there are only a few elephants. The key point is that counters corresponding to elephants will not be set to 0. This property is important if one wants to accurately estimate the number of packets of elephants.

Two different criteria to declare when the state of the filter is overloaded are proposed.

— RATIO criterion. Define r as the proportion of non null counters in the multistage filter, the filter is overloaded when r is above some threshold R (90% for example).

— AVERAGE criterion. Define avg as the average of counters values. The filter is overloaded when avg is above some threshold AVG ($C/2$ for example).

The adaptive property of the scheme proposed is clear: As long as the state of the filters is not overloaded then nothing occurs and if there is a peak of activity, quite quickly the filters will be filled and the refreshment mechanism will be automatically executed.

The rationale behind the RATIO criterion is that if most of counters are non-zero then very likely mice have contributed to a significant fraction of the values of the counters so that false elephants will show up. While usually most of mice comprise 1 or 2 packets, it may happen that, for some traffic types, their sizes may be larger so that if a few mice are mapped to the same entry, it is sufficient to increment the corresponding counter by 1. This is why the AVERAGE condition considers the average value of counters rather than the number of non-zero counters. Our experiments show nevertheless that condition RATIO is sufficient for most of IP traffic types.

Because the number of mice is much larger than the number of elephants, collisions between elephants and mice can be neglected. False elephants are mainly caused by collisions in the hash table between short flows. Missing elephants is the drawback of the algorithm. An elephant having f packets, $f \geq C$, can be missed if its counters do not reach the threshold C because of the refreshment mechanism (all counters are decreased by one when the state of the filters is overloaded).

B. Active Elephants and Statistical Characteristics

The number of entries in the memory storing elephants gives as estimation of their total number. It is also possible to store additional variables for each flow in this memory, for instance the starting and finishing time of the elephant corresponding to the arrival times of the first and last packets, respectively, the number of packets, the total volume in bytes, the number of segments of a certain types (typically SYN segments for attack detection), etc.

C. An Improvement: the virtual filter

Missed elephants can be divided into two categories: elephants with low throughput (less than the refreshment frequency) and small elephants. An elephant having a number of packets slightly larger than C, can then be missed if there is at least one refreshment during its life time. The following improvement of the algorithm aims at reducing the number of missed elephants by giving small elephants more chance to be captured.

The available memory is divided in two halves. In the first half, a Bloom filter as defined above is implemented, it will be called virtual filter. It operates exactly in the same way for incrementing and refreshing counters. The second half is another Bloom filter, called the real filter; its counters are incremented in the same way as for the virtual filter but no refreshment mechanism is used except that when a counter becomes equal to 0 in the virtual filter, in that case, it is also set to 0 too.

The proportion of non null counters is the same for the two filters. The identification of elephants is done with the values of counters of the real filter, when all the counters corresponding to some flow are equal to C. Note that since the counters are not decremented by one, it is less likely that some packets of elephants will be lost in this manner. The value of a counter in the real filter is therefore always higher than (or equal to) the corresponding counter in the virtual filter. The number of identified elephants is thus higher than what we obtain with the initial version of the algorithm. In particular small elephants have more chance of being identified.

The drawback of the virtual filter is that, in some cases, it can introduce some new false positives. In fact, as the counters in the real filter are now higher, mice are more likely to be considered as elephants. This especially happens when the mean size of mice is not small enough compared to the threshold C.

III. Experimental Results

In this section, the efficiency of the algorithm and the impact of some of its parameters are discussed. Another more extensive set of experiments can be found in Azzana [6], for the impact of virtual filter in particular.

To evaluate the performance of the algorithm, two different traces have been tested: the first trace contains commercial traffic from the France Telecom IP backbone network carrying ADSL traffic. This traffic trace has been captured on a Gigabit Ethernet link in October 2003 between 9:00 pm and 10:00 pm. This period corresponding to a peak activity by ADSL customers, its duration is 1 hour and contains more than 10 millions of TCP flows. The second trace “20040601-193121-1”, URL: http://pma.nlanr.net/Traces/Traces/long/ipls/3/, contains academic traffic issued from Abilene III.

A. Results

In our experiments, the filter consists of 10 stages associated to 10 independent random hash functions ($k = 10$). Elephants are here defined as flows with at least 20 packets ($C = 20$).

First we apply the algorithm proposed by Estan and Varghese [5] to the France Telecom trace in order to identify elephants. Recall that this algorithm uses a periodic reinitialization to zero of all counters to refresh the filter. Results

are compared to the adaptive refreshment using the RATIO criterion. To be fair in the comparison, at a refreshment instant, instead of decrementing by one, all counters are set to zero like in Estan and Varghese algorithm. The refreshment time period is set to 5 seconds.

The number of new elephants per minute found by the algorithms and its exact value are plotted in Figure 2. It shows that the periodic refreshment of Estan and Varghese (5 seconds) is not adapted to the traffic trace since many elephants are missed in this case. The refreshment frequency is too high and elephants cannot send their 20 packets in only 5 seconds. This is due to the fact that in the ADSL traffic trace, elephants are generated by peer to peer file transfers, which are basically with low bit rates (see Ben Azzouna *et al.* [9] for more details).

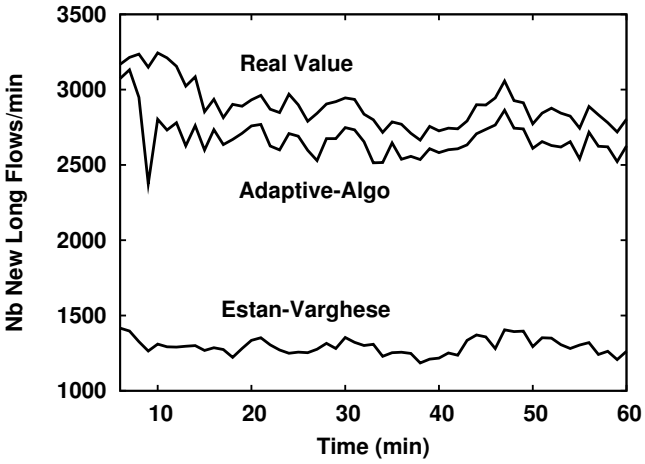

Fig. 2. Impact of the adaptive refreshment on the estimation of elephants number, memory $1.31MB$, $R = 90\%$, France Telecom trace

A change of the value of the period would probably improve the accuracy but it is not clear how it can be done "on line". On a one hour long traffic trace, this parameter has to be changed regularly. This is not necessary for short traces (say, a few tens of thousands of packets) used in Estan and Varghese [5], but this becomes an issue for long traffic traces. Using the adaptive refreshment with a threshold $R = 90\%$ and a small memory of $1.31MB$, only about 12% of the elephants are missed. With a memory size of 5.24MB, the error is of the order of 2%. See Figure 7 below.

Another important feature of the adaptive algorithm which can be seen from Figure 2 is that it follows quite closely the variations of elephant traffic, this is also true for Estan and Varghese algorithm but in a much less accurate way. This is, in our view, the benefit of the adaptive property of our algorithm.

Figure 3 gives the relative error on the estimation of elephants number for the three versions of the algorithm: with the refreshment using RATIO and AVERAGE criteria. Both RATIO and AVERAGE criteria give accurate estimations of the total number of elephants. The fact that the relative error remains under 7% for all the duration of the trace shows stability and robustness of the algorithm. The same experiments performed on Abilene trace give similar results. See Figure 4. So the adaptive algorithm is an efficient method to refresh the filter without impacting too much the estimation of the number of elephants.

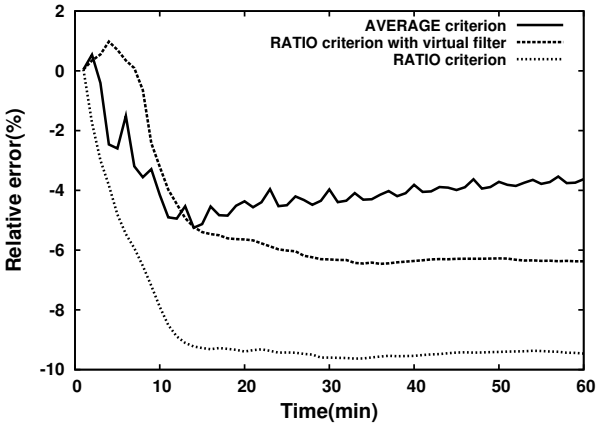

Fig. 3. Impact of refreshing mechanism and the virtual filter on the estimation of elephants number, $M = 1.31MB$, $R = 90\%$, France Telecom trace

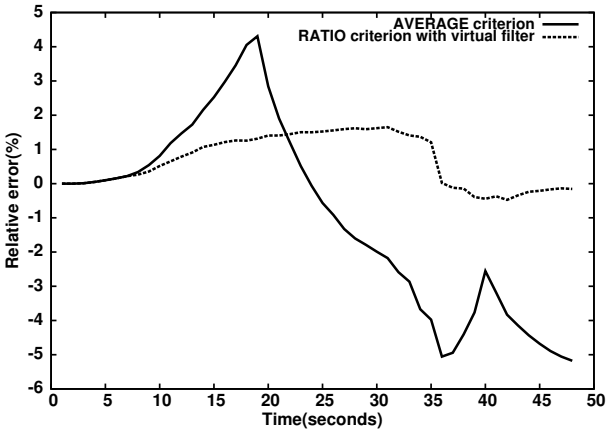

Fig. 4. Comparison of refreshing criteria for Abilene trace with M=$1.31MB$ and $R = 90\%$.

Once an elephant has been identified, it is registered in an auxiliary memory together with the number of packets seen and each time a packet of this flow is seen, this value is incremented by 1. In this way, one can estimate the statistics of the sizes of elephants. Figures 5 and 6 show that this statistics of this estimation of the number of packets per elephant is really very close to the real value for the two different traces.

B. Impact of the M and R parameters

In Figure 7, we analyze the impact of the size M of the memory used for the Bloom filter on the estimation of the number of the elephants. As expected, using a larger memory improves the estimation. The error is very close to zero with a memory size of only $5MB$. In fact the filter is refreshed less frequently which gives more chance for elephants to be detected.

Figure 8 shows the dependence of the accuracy of the estimate for several values of the threshold R. A threshold of 90% gives a good estimation of the number of elephants. We just miss about 7% of the elephants. With a higher threshold, we miss less elephants but some false positives can

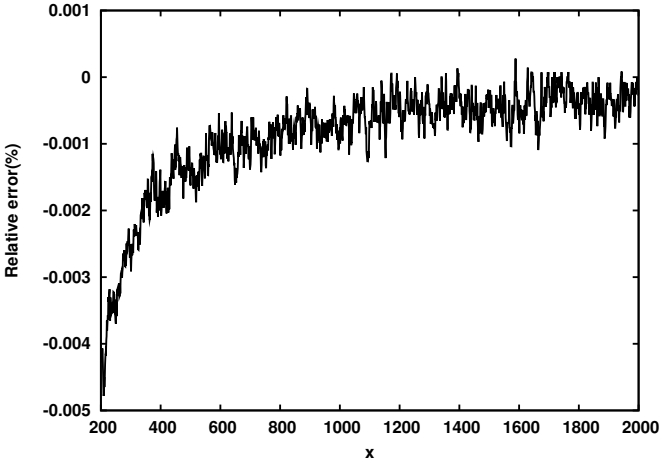

Fig. 5. Relative error on the number of flows with more than x packets, $M = 1.31$MB, $R = 90\%$. France Telecom trace

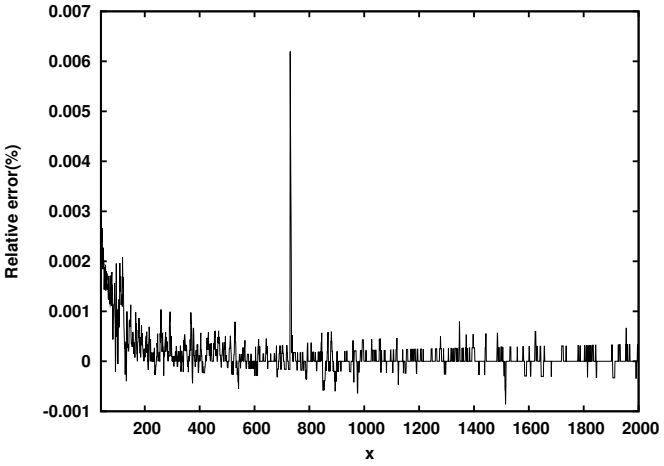

Fig. 6. Relative error on the number of flows with more than x packets, $M = 1.31MB$, $R = 90\%$. Abilene trace

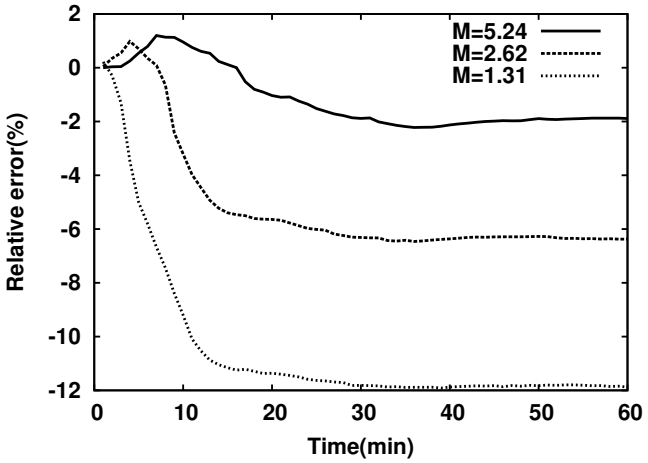

Fig. 7. Impact of memory size M of the filter, $R = 90\%$

be added. So there is clearly a trade-off on the choice of R. See Chabchoub *et al.* [10] for more details.

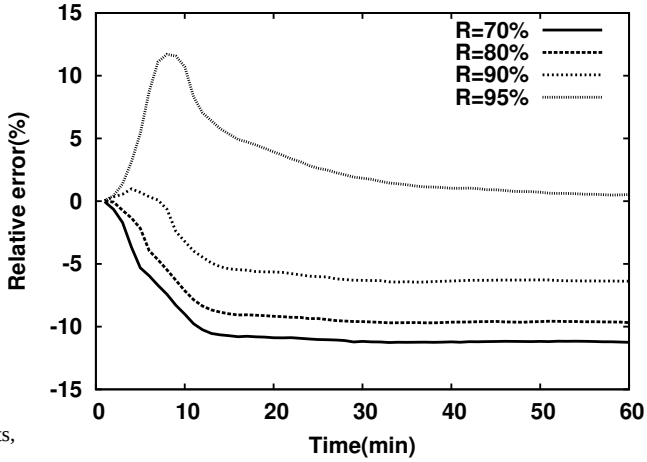

Fig. 8. Impact of R for RATIO criterion, $M = 1.31Mo$

IV. Application: Attacks Detection

A. Context

The algorithm proposed in Section II can be adapted to various purposes, notably attack detection in the Internet. We focus here on a particular kind of attacks referred to as denial of service (DoS) attacks in the technical literature. Here we are only interested in *SYN flood* and *volume flood* which are the most common DoS attacks. See Hussain *et al.* [11] for a classification of DoS attacks.

A *SYN flood* exploits a weakness in the connection phase of TCP, also called "the three way handshake". This attack consists of sending a large number of SYN packets to the same destination (or group of destinations) during a small interval of time. Due to the TCP implementation, the destination allocates resources to all these connection requests and will maintain many half-open connections waiting for acknowledgments from sources for about one minute. A large number of SYN packets consume therefore a significant fraction of the resources of the targets and, in the end, the corresponding machines become unreachable (see Wang *et al.* [12] for more details). In this setting the goal is to design an one-line algorithm which can detect an attack in less than a minute. Such a detection can be used by the network operator in order to filter SYN segments towards the victim.

While a SYN flood consists of a sudden arrival of a large number of SYN segments, a *volume flood* attack uses a few TCP flows and gradually transmits with a steady increase of the transmission rate a huge amount of data which will consume the available bandwidth of the target.

Several methods have been developed in the technical literature for DoS detection; they are mainly based on TCP properties such as periodicity in Chang *et al.* [13], or SYN and FIN packets counting in Wang *et al.* [12], Barford *et al.* [14], Krishnamurty *et al.* [15]. Most of them suffer from scalability or robustness especially if only sampled traffic is available as it is usually the case in backbone networks.

For SYN flood detection, the main difficulty is in distinguishing between the (normal) variations of traffic and a sudden, anomalous sequence of SYN packets. In the literature attacks are sometimes defined as a notable variation from the standard behavior of specific parameters of the model used: parameters of the specific statistical models or long range dependence variables for signal processing description of traffic for example. If the algorithms based on these representations may be efficient to detect some anomalous behaviors, they cannot, in general, assert the nature of the attack because they handle traffic aggregates without distinction of flows Chatelain *et al.* [16] and Lakhina *et al.* [17].

B. Adaptive scheme to detect SYN flood attacks

The algorithm proposed for an on-line detection of SYN and volume flood is mainly based on the algorithm detailed in Section II, but with a different refreshing mechanism of the Bloom filter.

As explained above, SYN packets with the same destination address are aggregated as a single "flow". In this case, by using a Bloom filter as before, the refreshing mechanism of the multistage filter has a different purpose: it should eliminate quickly all normal flows using an aggressive refreshing mechanism so that if a "long" flow survives then it must be a SYN flood attack. As it is easily seen, the term "long" has to be properly defined. Roughly speaking, this means that such a flow is much longer than the other "normal" flows. Again, because of the variation of traffic, an adaptive scheme has to be devised to define properly these concepts.

The main idea of the algorithm is to evaluate a varying average m_n of the largest flow in several sliding time windows of length Δ. The quantity m_n describes "normal flows"; it is periodically updated in order to adapt to varying traffic conditions. It is a weighted average that takes into account all its past values to follow carefully traffic variations but not too closely. If a flow in the nth time window is much larger than m_{n-1}, it is considered as an attack, and the moving average is not updated for this time window.

The following variables are used.

- As before, r is the proportion of non-zero counters in the Bloom filter.
- S is a multiplicative detection threshold. Roughly speaking, an attack is declared when an observation is S times greater than the "normal" behavior. It is determined by the administrator.
- R_s and R are thresholds for the variable r. The constant R_s is independent of traffic and taken once for all equal to 50% and R is a variable threshold depending on the traffic type considered.
- α is the updating coefficient for averages, 0.85 in our experiments.
- Δ duration of the initialization phase (1mn in this paper).
- m_n is the weighted moving average in the nth time window.

The algorithm starts with an initialization phase of length Δ in order to evaluate the threshold R. At the end of this phase, R will be definitively fixed for the rest of the experiment. In addition, as this phase corresponds to the first time window, the moving average m_1 will be initialized as the biggest counter obtained. See Table I for the description of the algorithm.

Initialization phase:
- All counters are 0.
- The Bloom filter is progressively updated with SYN packets by using their destination address.
  - After a duration Δ, evaluate the variable r
    - if $r \leq R_s$ then $R := r$ else $R := R_s$.
    - $m_1 :=$ maximum of the values of counters of the multistage filter.

Detection phase: the $n\ th$ time window
- At the beginning all counters are initialized to 0.
- The Bloom filter is progressively updated with SYN packets by using their destination address.
  - if a counter exceeds $S\ m_{n-1}$, an attack is declared.
  - if $r \geq R$
    - $\max_n$: maximum of the values of counters of the multistage filter.
    - if $\max_n < S\ m_{n-1}$ $m_n = \alpha\ m_{n-1} + (1-\alpha) \max_n$
    - start the $(n+1)$th time window.

TABLE I
ALGORITHM FOR SYN FLOOD DETECTION.

Note that an alarm is declared during the nth time window when the value of a counter is greater than Sm_n. At the beginning, the first time window is fixed (its duration is Δ) but, since the evolution depends on the occupation rate of the filters, the duration of the other time windows will be variable. If traffic characteristics are not much varying, time windows durations will be around one minute. In this case, an attack will be detected at the latest after one minute so that the network administrator can react quickly.

C. Detection of volume flood attacks

For progressive attacks, the impact on the traffic cannot be clearly seen in a time window of one minute. In fact the attack can be so slow that it could be locally considered as a normal traffic variation. This kind of attacks has typically a long duration. In this situation, we should consider a larger time window in order to detect the anomalous impact of the attack on traffic. Thus, to cope with these attacks, one consider a similar algorithm but with a larger time window Δ' of 5 minutes. This new filter operates in the same way but on a longer time scale and is completely independent of the first filter. In particular, it has its own parameters: R', r', R'_s, m'_n, $\max'_n$, and S'.

D. Experimental Results

To evaluate and validate the attack detection algorithm described in the previous section, we run experiments with two France Telecom traces, one from the IP collect network carrying in majority ADSL traffic and the other form the IP transit network (OTIP). In this latter case, only sampled traffic is available. The characteristics of the traffic traces are given in Table II.

To detect SYN and volume flood using two time scales ($\Delta = 1$mn and $\Delta' = 5$mn), we need four filters. Each filter

Traces	Nb. sampled IP packets	Nb. Flows sampled	Duration
OTIP	105.10^6	4.10^6	3 days
ADSL	825.10^5	32.10^5	3 hours

TABLE II
CHARACTERISTICS OF TRAFFIC TRACES USED FOR ATTACK DETECTION.

contains ten stages ($k = 10$) and has a total size M around $1MB$.

In Figure 9, the ADSL trace is divided into several time windows of 5mn and, for each interval, the volume of the largest SYN flow is computed. The observed peaks seem to correspond to attacks. Tested on this trace, the algorithm detected two SYN flood against two different IP addresses. The response time of the algorithm is satisfactory as the alarms are raised at the beginning of the attacks. It should be noted that when the duration of the time window is 1mn, only the second attack is detected.

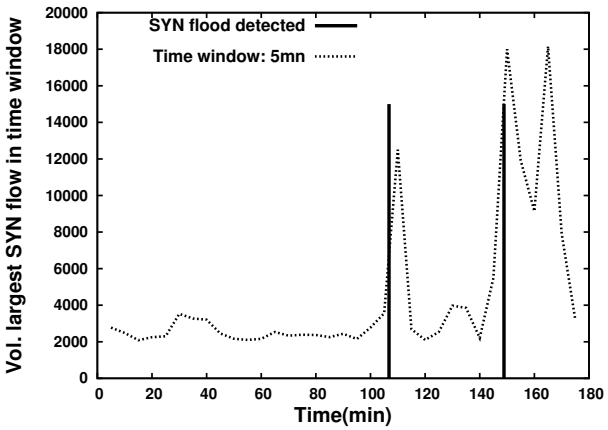

Fig. 9. SYN flood detection for ADSL trace with S=5 and S'=3

In Figure 10, the same trace is used to detect volume flood. The volume of the flow is now the number of packets which are not SYN packets. SYN packets are not computed to prevent from considering some SYN flood as volume flood. The algorithm detects one volume flood using the time window of 5mn. When the duration of the time window 1mn, no attack is detected.

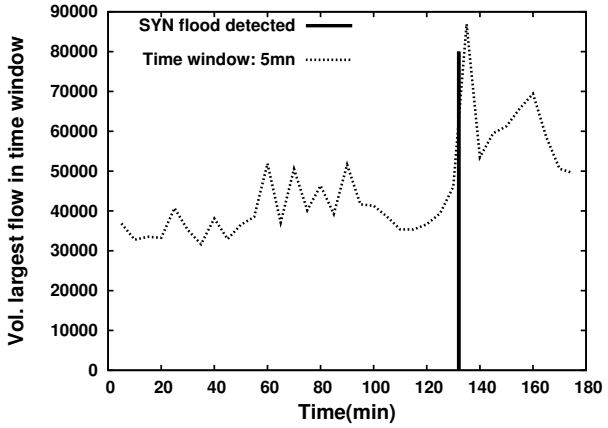

Fig. 10. Volume flood detection for ADSL trace with S=5 and S'=2

In Figures 11 and 12 the OTIP trace is considered. This trace contains many attacks. The algorithm raises several alarms which coincide with the largest flows represented by the highest peaks.

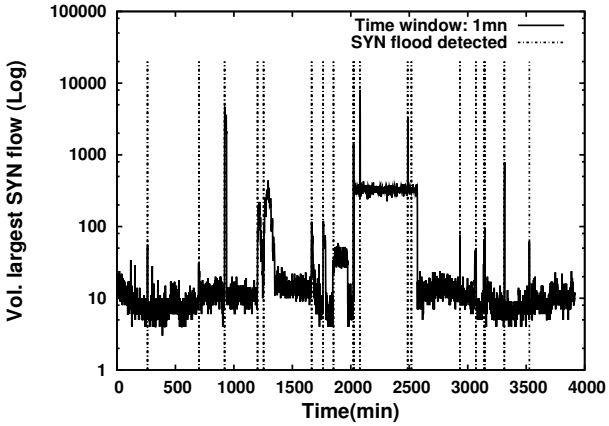

Fig. 11. SYN flood detection for OTIP trace with $S = 5$ and $S' = 3$

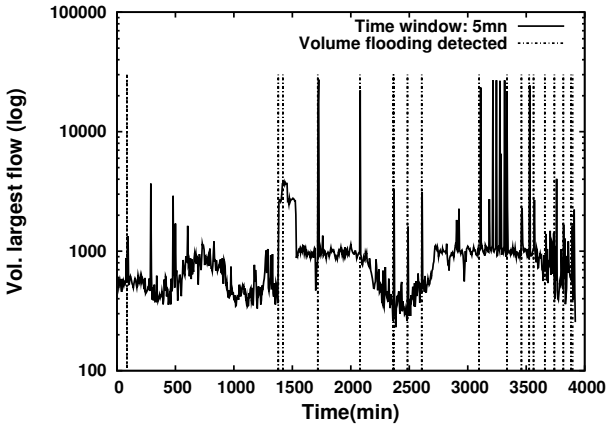

Fig. 12. Volume flood detection for OTIP trace with $S = 5$ and $S' = 2$

V. PERFORMANCE ISSUES

To evaluate the performance of the algorithm identifying elephants, one can estimate the error generated by this algorithm in terms of the different involved parameters:

- R: overload ratio of the filter,
- C: minimum size of large flows (elephants),
- m: number of counters per hash table,
- k: number of hash tables in the filter

In Chabchoub *et al.* [10], a theoretical analysis of the version of the algorithm using the RATIO criterion is proposed. The aim is to estimate the error generated by mice, i.e., the ratio of false positives. Simplified models have been investigated for that purpose. In a first step, a one-stage filter is considered and the traffic is supposed to be made up only of mice of size 1.

The analysis uses Markovian techniques: If $W_n^m(i)$ is defined as the proportion of counters with values i, $i \in 0, \ldots, C$ just before the nth refreshment for a filter with m counters then the process $(W_n^m, n \geq 0) = ((W_n^m(i), i = 0, \ldots, C), n \geq 0)$ is a Markov chain on a finite state space with invariant

distribution π_m. As m gets large, it is shown that the sequence $(W_n^m, n \geq 1)$ converges to a dynamical system such that the limit $\overline{w}$ of the sequence (π_m) is its unique fixed point. It turns out that the limiting distribution $\overline{w}$ has a nice interpretation in terms of the stationary measure μ_λ of a $M/G/1/C$ queue with service time 1 where the arrival rate λ depends on μ_λ. This allows us to compute the invariant measure as a solution of a linear system of $C+1$ equations and $\lambda(\overline{w})$ is the solution of a fixed point equation. At equilibrium, the average number of packets between two refreshing instants is of the order of $\lambda(\overline{w})$. Due to finite capacity C this quantity is greater than the number of removed packets at each refreshment. In particular $\lambda(\overline{w})$ is not necessarily less than 1. For R enough close to 1, it is shown that $\lambda(\overline{w})$ can in fact exceed 1 which changes the qualitative behavior of the system: if the arrival rate $\lambda(\overline{w})$ is less than 1, then $\overline{w}$ is concentrated on small values $0,\ 1,\ 2\ldots$ of the state space $\{0, 1, \ldots C\}$. When $\lambda(\overline{w}) > 1$ the distribution $\overline{w}$ is mainly concentrated on the values C, $C-1, \ldots$ since false positives are closely related to the quantity $\overline{w}(C)$, this implies that the proportion of false positives is much higher in this case. Consequently, there is a critical value $r_c < 1$ of R for which $\lambda(\overline{w}) = 1$ so that the algorithm does not perform well when $R > r_C$.

Using k stages, the probability that a mouse gives rise to a false positive is less than $\overline{w}(C)^k$. When mice have general size distribution, this model is extended to an approximated model where packets of a given mouse arrive simultaneously. In this case a $M/G/1/C$ queue with batch arrivals with distribution equal to the mouse size distribution plays an important role. The corresponding critical value r_c in this case can be expressed numerically.

VI. Concluding remarks

We have presented in this paper an original adaptive algorithm for identifying elephants in Internet traffic. As earlier proposed by Estan and Varghese, this algorithm is based on Bloom filters, but instead of periodically erasing the filter, we introduce different criteria to decrement the various counters of the filter. In order to improve the accuracy of the algorithm, we have introduced the concept of virtual filter, whose counters are less frequently decreased. The proposed algorithm has been tested against different traffic traces and performs better than the one by Estan and Varghese.

Finally, the proposed elephant counting mechanism has been adapted in order to detect flood attacks (SYN and volume floods) in Internet traffic. This gives rise to a new algorithm, whose key parameters adapt to network traffic. This algorithm has been successfully tested when considering two types of traffic traces (corresponding to residential traffic and transit traffic).

References

[1] K. Papagiannaki, N. Taft, S. Bhattacharyya, P. Thiran, K. Salamatian, and C. Diot, "A pragmatic definition of elephants in internet backbone traffic," in *Internet Measurement Workshop*. ACM, 2002, pp. 175–176. [Online]. Available: http://doi.acm.org/10.1145/637201.637227

[2] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270–313, 2003.

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

[4] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[5] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. Sigcomm'02*, Pittsburgh, Pennsylvania, USA, August 19-23 2002.

[6] Y. Azzana, "Mesures de la topologie et du trafic internet," Ph.D. dissertation, Université de Paris 6, July 2006. [Online]. Available: http://www-c.inria.fr/twiki/pub/RAP/FormerMembers/Azzana-PhD.pdf

[7] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," in *Proceedings of the 13th conference on analysis of algorithm (AofA 07)*, 2007, pp. 127–146.

[8] F. Giroire and E. Fusy, "Estimating the number of active flows in a data stream over a sliding window," in *Proceedings of the Fourth Workshop on Analytic Algorithmics and Combinatorics (ANALC0)*, D. Applegate, Ed. New Orleans: SIAM, Jan. 2007, pp. 223–231.

[9] N. B. Azzouna, F. Clérot, C. Fricker, and F. Guillemin, "A flow-based approach to modeling ADSL traffic on an IP backbone link," *Annals of Telecommunications*, vol. 59, no. 11-12, pp. 1260–1299, November-December 2004.

[10] Y. Chabchoub, C. Fricker, F. Meunier, and D. Tibi, "Analysis of an algorithm catching elephants on the internet," in *Fifth Colloquium on Mathematics and Computer Science*, ser. DMTCS Proceedings Series, september 2008, pp. 299–314.

[11] A. Hussain, J. Heidmann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *ACM/SIGCOMM*, Aug. 2003.

[12] H. Wang, D. Zhang, and K. Shin, "Detecting syn flooding attacks," in *IEEE Infocom'02*, 2002.

[13] C. Cheng, T. Kung, and K. Tan, "Use of spectral analysis in defense against dos attaks," in *IEEE Globecom'02*, 2002.

[14] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *ACM/SIGCOMM IMW*, 2002.

[15] B. Krishnamurty, S. Sen, Y. Zhang, and Y. Chen, "Sketch based change detection : Methods, evaluation, and applications," in *ACM IMC*, 2003.

[16] F. Chatelain, P. Borgnat, J.-Y. Tourneret, and P. Abry, "Parameter estimation for sums of correlated gamma random variables. Application to anomaly detection in internet traffic," in *Proc IEEE Int. Conf. on Acoust., Speech and Signal Proc. ICASSP-08,*, Las Vegas (NV), 2008.

[17] A. Lakhina, M. Crovella, and C. Diot., "Detecting distributed attacks using network-wide flow data," in *Proc. FloCon 2005 Analysis Workshop*, New Orleans, September 2005.